ALEXANDER TEICHER

Entwurf und Implementierung einer webbasierten Datenbank für autosomale STR-Marker

Diplomarbeit

HOCHSCHULE MITTWEIDA

UNIVERSITY OF APPLIED SCIENCES

Fakultät

Elektro- und Informationstechnik

Mittweida, 2010

ALEXANDER TEICHER
Matrikelnummer: 18689

Entwurf und Implementierung einer webbasierten Datenbank für autosomale STR-Marker

eingereicht als

Diplomarbeit

an der

HOCHSCHULE MITTWEIDA

UNIVERSITY OF APPLIED SCIENCES

Fakultät

Elektro- und Informationstechnik

Mittweida, 2010

Erstprüfer: Prof. Dr. rer. nat. Dirk Labudde
Zweitprüfer: Dr. rer. nat. Frank Götz

Vorgelegte Arbeit wurde verteidigt am: 16. Dezember 2010

Bibliographische Beschreibung:

Alexander Teicher:

Entwurf und Implementierung einer webbasierten Datenbank für autosomale STR-Marker 2010. - 88 S. Mittweida,

Hochschule Mittweida - University of Applied Sciences,

Fakultät Elektro- und Informationstechnik, Diplomarbeit, 2010

Referat:

Die Verwendung von Short tandem repeats (STRs) zur Durchführung von Vaterschaftsgutachten gilt mittlerweile als Standard in der forensischen Medizin. Diese DNA-Polymorphismen lassen sich jedoch nicht nur zur Bearbeitung von Abstammungsgutachten oder Spurenfällen nutzen, sondern finden auch Verwendung in der Populationsgenetik. Während der wissenschaftlichen Arbeit soll eine webbasierte Datenbank für die Speicherung dieser Daten entstehen.

Danksagung

Die vorliegende Diplomarbeit bildet den Abschluss meines Studiums der Multimediatechnik an der Hochschule Mittweida.

An dieser Stelle möchte ich mich bei Herrn Prof. Dr. rer. nat. Dirk Labudde und Herrn Dr. rer. nat. Frank Götz für die hervorragende Betreuung sowie dem entgegengebrachten Vertrauen während der gesamten Bearbeitungszeit bedanken.

Ferner danke ich den restlichen Mitarbeitern der Qualitype AG für die technische Unterstützung sowie für die vielen Anregungen und Hinweise, die zum Gelingen dieser Arbeit beigetragen haben.

Auch gilt mein Dank meiner Freundin Carolin Küntzel, die sich opferte und als unbeteiligte Lektorin zur Verfügung gestellt hat und mir über kleine Durststrecken hinweg half sowie Jens Schröter der mich ebenfalls als Lektor unterstützte.

Mittweida, den 31. Oktober 2010 Alexander Teicher

Inhalt

Abbildungsverzeichnis

Tabellenverzeichnis

Formelverzeichnis

Abkürzungsverzeichnis

AJAX	Asynchronous Java Script and XML
AOP	Aspektorientierte Programmierung
API	Application Programming Interface
AS	Application Server
BMP	Bean Managed Persistence
CMP	Container Managed Persistence
CSS	Cascading Style Sheets
CRUD	Create, Read, Update und Delete
CSV	Comma-Separated Values
DI	Dependency Injection
DNA	Deoxyribonucleic acid
DNS	Desoxidribonukleinsäure
DRY-Konzept	Don´t repeat yourself-Konzept
EJB	Enterprise JavaBeans
EL	Expression Language
h	Homozygotie
HET	Heterozygotie
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ID	Identifier
IDE	Integrated Development Environment
IoC	Inversion of Control
ISFG	International Society for Forensic Genetics
JAR	Java Archive

Java EE	Java Enterprise Edition
Java SE	Java Second Edition
jBPM	Java Business Process Management
JDBC	Java Database Connectivity
JPDL	jBPM Process Definition Language
JPA	Java Persistence API
JRE	Java Runtime Environment
JSF	Java ServerFaces
JSP	Java ServerPages
MDB	Message Driven Bean
MEC	Mean Exclusion Chance
MVC-Modell	Model-View-Controller Modell
ORM	Object-Relational Mapping
PCR	Polymerase-Kettenraktion
PD	Power of Discrimination
PDA	Personal Digital Assistent
PDF	Portable Document Format
PE	Power of Exclusion
PHP	Hypertext Preprocessor
PI	Paternity Index
PIC	Polimorphic Information Content
RDBMS	Relational Database Management System
RIA	Rich Internet Application
STRs	Short Tandem Repeats
SQL	Structured Query Language

TCP/IP	Transmission Control Protocol und Internet Protocol
UI	User Interface
UML	Unified Modeling Language
URL	Uniform Resource Locator
WWW	World Wide Web
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language

1 Einleitung

In der vorliegenden wissenschaftlichen Arbeit wird das Thema „Entwurf und Implementierung einer webbasierten Datenbank für autosomale STR-Marker“ vorgestellt und bearbeitet. Das Thema wird kurz in diesem Kapitel einführend beschrieben, bevor die Zielsetzungen der Diplomarbeit sowie deren Aufbau dargestellt werden.

1.1 Einführung in die Thematik

Heutzutage spielt die Erstellung molekulargenetischer Gutachten unter Verwendung populationsgenetischer Daten in der Rechtsmedizin eine wichtige Rolle. Dabei sind unter anderen die Auswahl und Herkunft der verwendeten Daten von großer Bedeutung, nicht nur für das biostatistische Ergebnis, sondern besonders auch für die anschließende Bewertung. Dies erklärt den Zweck großer genetischer Datenbanken als Quelle breitgefächerter und fundierter Informationen.

1.2 Zielsetzung dieser Diplomarbeit

Die Einführung in die Thematik soll zeigen, dass ein großer Bedarf an einer solchen Datenbank besteht. Es gibt bereits Projekte zum Aufbau einer solchen Datenbank. In der hier vorliegenden Arbeit sollen verschiedene, bestehende Systeme untersucht werden. Aus den daraus gesammelten Erkenntnissen sowie selbst eingebrachten Wissensgrundlagen soll ein Entwurf als Grundlage für eine Implementierung eines Prototyps definiert werden. Im Anschluss daran soll dieser Entwurf unter dem Einsatz von *Java Enterprise Edition*-Technologien umgesetzt werden. Als Stammdaten sollen Datensätze aus der *DNA*-Analyse-Software *GenoProof®2* dienen.
Wichtig bei dieser wissenschaftlichen Arbeit ist es, das Problem aus dem Gebiet der Bioinformatik vollständig zu erfassen und zu verstehen und mit Hilfe der gesammelten Kenntnisse ein System für diese Thematik zu entwickeln, welches die Verwaltung von Populationsdaten wesentlich vereinfacht und ein wirksames Werkzeug für die forensische Gemeinschaft darstellt.

1.3 Kapitelübersicht zur Diplomarbeit

In **Kapitel 2** werden biologische Grundlagen erläutert, in welche die Diplomarbeit eingebettet ist. Weiterhin werden bestehende Datenbanken beleuchtet und ein Anforderungskatalog formuliert.

Kapitel 3 beschäftigt sich mit den Grundlagen der Informatik die mit der Diplomarbeit in Verbindung stehen. Dabei werden verschiedene Technologien wie z.B. *Java Enterprise Edition*, *Asynchronous Java And XML*, sowie Architekturmodelle wie das *Model-View-Controller Modell* erläutert.

Kapitel 4 zeigt die Notwendigkeit zum Einsatz von Web-Frameworks bei der Entwicklung webbasierter Anwendungen. Dabei werden zwei Vertreter: JBoss *Seam* und *Spring MVC* näher untersucht und anschließend evaluiert.

Kapitel 5 setzt sich mit dem Entwurf der Anwendung auseinander. Dabei werden u.a. funktionale und nicht funktionale Anforderungen sowie eine Softwarearchitektur definiert.

In **Kapitel 6** wird die praktische Diplomaufgabe beschrieben: Die Implementierung einer webbasierten Datenbank für autosomale STR-Marker mit Hilfe des Frameworks JBoss *Seam*.

Kapitel 7 beschreibt zwei mögliche Anwendungsszenarien, die die möglichen Einsatzgebiete der Webanwendung aufzeigen.

Im **Kapitel 8** werden die Ergebnisse der Arbeit zusammengefasst und mögliche Ausblicke zur Weiterführung der Thematik vorgestellt.

2 Autosomale Datenbanken

2.1 Biologische Grundlagen

Zum Verständnis des biologischen Hintergrundes für diese Diplomarbeit, soll das nachfolgende Kapitel dem Leser das Verständnis zum Einsatz einer autosomalen Datenbank liefern. Des Weiteren bietet das Glossar eine Übersicht zu den biologischen Fachbegriffen. Die biologischen Grundlagen werden mit den biostatistischen Berechnungsgrundlagen abgeschlossen.

2.1.1 Biologischer Hintergrund

Die Voraussetzung zur Analyse des Erbmaterials ist zunächst die Extraktion der *Desoxyribonukleinsäure* (DNS, englisch *DNA*) aus dem Probenmaterial, wie Haaren, Mundschleimhaut, Hautzellen oder Körperflüssigkeiten (z.B. Blut oder Urin). Die *DNA* ist der Träger der Erbinformation und besteht aus den vier Basen Adenin, Guanin, Thymin und Cytosin. [1] An diesen Basen befinden sich weiterhin jeweils ein Zuckermolekül die *Desoxyribose*, sowie ein Phosphatrest von der Phosphorsäure. Zuckermolekül und Phosphatrest bilden dabei das Rückgrat des *DNA*-Stranges, wie es in Abbildung 2-1 zu sehen ist. [2]

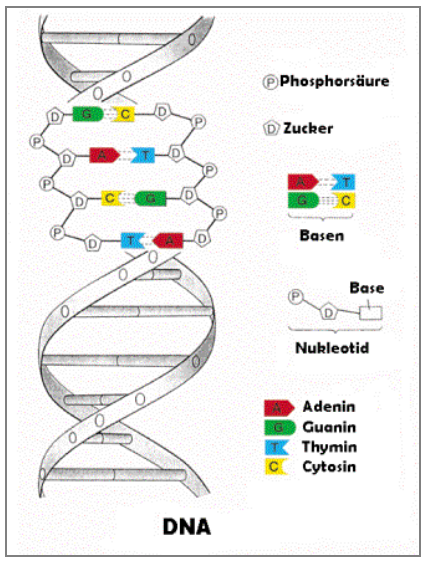


Abbildung 2-1: DNA – Grundaufbau einer doppelsträngigen DNA mit den komplementären Basen: Adenin-Thymin und Cytosin-Guanin, sowie dem Zucker und der Phosphorsäure. Basenpaar, Zucker und Phosphatrest bilden das Nukleotid [3]

Die *DNA* ist in Form von *Chromosomen* gespeichert, jedes einzelne *Chromosom* enthält dabei jeweils einen langen, kontinuierlichen *DNA*-Doppelstrang. *Chromosomen* sind lange, fadenförmige Gebilde, welche aus *DNA* und *Protein* bestehen, das sogenannte *Chromatin*. Sie kommen im Zellkern von *Eukaryoten* vor, darunter fallen Tiere, Pflanzen und Pilze. [4] Jede Tier- und Pflanzenart weist dabei eine unterschiedliche Anzahl an *Chromosomen* auf, diese Anzahl ist jedoch nicht äquivalent zu der Komplexität des Organismus. Dies verdeutlicht das Beispiel, dass der Mensch 46 *Chromosomen* und ein Goldfisch 94 *Chromosomen*, aufweist. [1] *Chromosomen* lassen sich in zwei Gruppen unterteilen, auf der einen Seite die autosomalen *Chromosomen* und auf der anderen Seite die Geschlechtschromosomen, dazu gehören die X- und Y-Chromosomen. Sie sind dafür verantwortlich welche Geschlechtszellen, männliche (XY) oder weibliche (XX), der Organismus ausbildet. [5]

Die auf dem *DNA*-Strang liegenden Basenpaare bilden den eindeutigen *Genotyp* (Erbbild) eines Individuums. [6] Für die weitere Analyse muss nicht der gesamte Strang untersucht werden. Es ist ausreichend nur bestimmte, charakteristische Regionen zu untersuchen. Diese werden als *Locus* bezeichnet (fortan *Marker* genannt). In diesen Regionen liegen sogenannte *Short Tandem Repeats* (*STRs*). Das sind kurze sich immer wiederholende Basenmuster. [6] In Abbildung 2-2 sieht man zwei *DNA*-Stränge mit den sich wiederholenden Basenmustern Adenin-Adenin-Thymin-Guanin (AATG). Auf dem oberen Strang gibt es sieben und beim unteren acht Wiederholungen des Basenmusters.

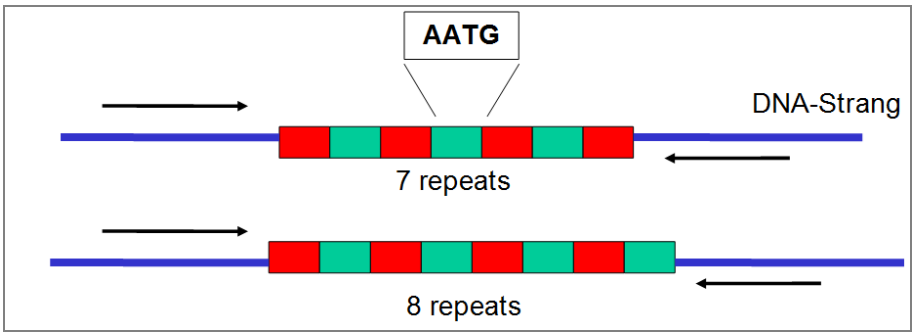


Abbildung 2-2: Short Tandem Repeats – kurze, wiederholende Basenmuster (Adenin-Adenin-Thymin-Guanin, kurz AATG), oberer DNA-Strang 7, unterer 8 Wiederholungen, diese Basenmuster dienen den bei der PCR verwendeten Primern als Zielregion, die Pfeile stellen die Arbeitsrichtungen der Polymerase dar [6]

Die Anzahl der Wiederholungen kann dabei bei verschiedenen Individuen stark variieren, so dass aus der Betrachtung mehrerer solcher *Marker*, ein individuelles Profil entsteht. [6] Diese charakteristischen *DNA*-Regionen, werden zunächst in der

Polymerase-Kettenraktion (*PCR*) mittels molekularbiologischen Methoden vervielfältigt. Nach der Aufspaltung des *DNA*-Doppelstrangs lokalisieren die in *Testkits* enthaltenen *Primer* die relevanten *Marker*. Anschließend werden sie kopiert, dabei werden Fragmente zur späteren Unterscheidbarkeit mit unterschiedlichen Fluoreszenzfarben markiert. Dieser Kopiervorgang wird mehrfach wiederholt und wird an beiden *DNA*-Einzelsträngen durchgeführt. Die entstehenden *DNA*-Fragmente vervielfältigen sich dabei exponentiell. Somit ist es möglich mit nur geringen Mengen von Spurenmaterial die Analyse und Weiterverarbeitung eines genetischen Profils durchzuführen. In einem *Sequenzierautomat* werden die erhaltenen *DNA*-Fragmente nun mittels *Gelelektrophorese* der Größe nach getrennt und mit einem Laser bestrahlt. Durch das Anlegen einer elektrischen Spannung wandern die geladenen Teilchen durch das Gel. Je größer diese Fragmente sind, desto langsamer bewegen sie sich (Trägheit). [7] Eine Kamera erfasst daraufhin die reflektierenden Fluoreszenzfarben der Fragmente und stellt die gemessenen Werte in einem bestimmten Ausgabeformat für die weitere Untersuchung zur Verfügung. Abbildung 2-3 zeigt die Auswertung einer *Gelelektrophorese*. [8]

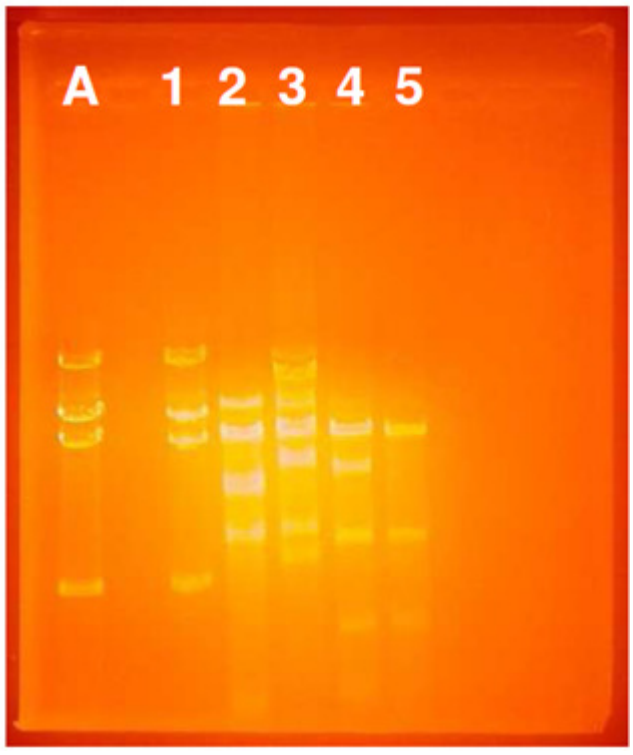


Abbildung 2-3: Gelelektrophorese – Auswertung einer Geleketrophorese mit 6 Proben, man kann schlussfolgern das Probe 1 ein ähnliches Muster wie Probe A aufweist [8]

Diese gesammelten Informationen stellen die Basis für die *DNA*-Analyse-Software (z.B. *GenoProof®2*). [9] Hier werden die *DNA*-Fragmente nach den spezifischen Anforderungen der Einsatzumgebung (Bestimmung der Länge, der Basenfolge, der Anzahl an Musterwiederholungen) verarbeitet und das individuelle genetische Profil erzeugt. In Abbildung 2-4 ist das individuelle Profil mit den untersuchten *Markern* (AM, D8S1179, SE33, etc.), eines Individuums zu sehen. Diese werden hier durch das Softwareprodukt *GenoProof®2* visualisiert.

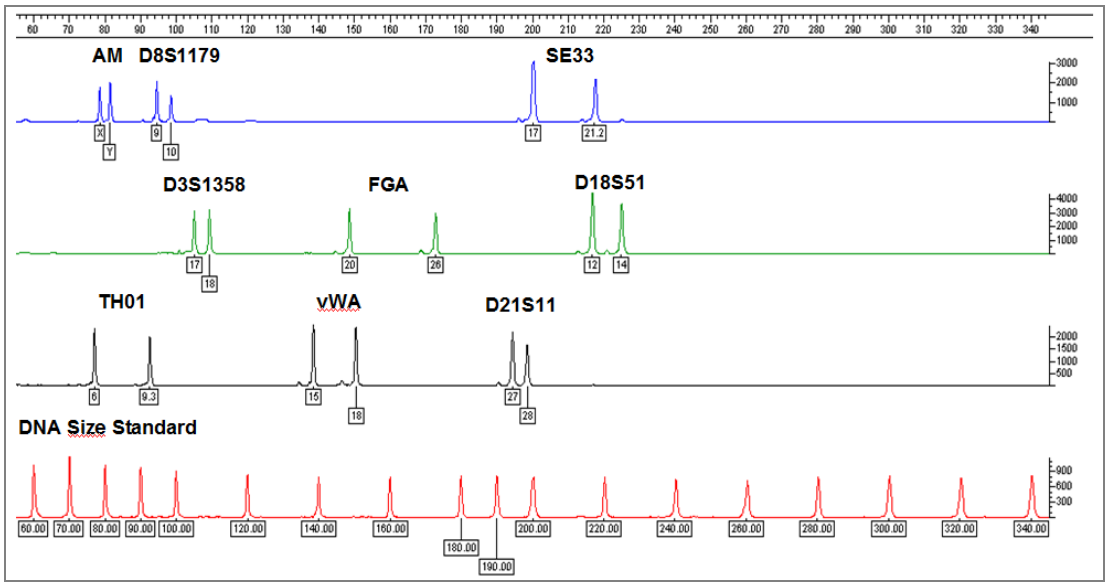


Abbildung 2-4: Genetisches Profil – Genetisches Profil eines Individuums, untersuchte Marker: AM, D8S1179, SE33, D3S1358, FGA, D18S51, TH01, vWA, D21S11, erstellt mit GenoProof®2 (DNA-Analyse-Software) [6]

Die statistische Untersuchung dieser genetischen Profile ist interessant für forensische Fragestellungen, aber auch für populationsgenetische Untersuchungen zu einem wertvollen Werkzeug geworden. Ein weiterer wichtiger Aspekt betrifft die Qualität der Vergleichsdaten, die die Grundlage für Wahrscheinlichkeitsberechnungen stellen. Es sind nur dann Aussagen zu der Häufigkeit der Merkmale eines Einzelindividuums zu treffen, wenn abgesicherte Kenntnisse über die tatsächliche Häufigkeit in der Grundgesamtheit vorliegen. [10] Der nachfolgende Absatz soll einige biostatistische Berechnungsgrundlagen näher beleuchten.

2.1.2 Biostatistische Berechnungsgrundlagen

Biostatistische Berechnungen erweitern die Funktionspalette einer autosomalen Datenbank enorm und bilden die Grundlage für die Qualitätssicherung der Daten. Aus

diesem Grund wurden sie in die autosomale Datenbank implementiert. In diesem Teil werden einige der im Prototyp zum Einsatz kommenden Berechnungsparameter mit den biologischen Hintergründen näher beleuchtet. Grundlage für diese Erkenntnisse bildete dabei u.a. das Theoriehandbuch zu *GenoProof®2*. [14]

Homozygotie (h) / Heterozygotie (HET)

Homozygotie (Reinerbigkeit) und *Heterozygotie* (Mischerbigkeit) sind Fachbegriffe aus der Genetik. [11] [12] Die meisten Organismen (z.B. Menschen) besitzen von jedem *Gen* (Abschnitt auf dem *DNA*-Strang), was z.B. die Blutgruppe oder Augenfarbe kodiert, zwei Kopien, im Normalfall eine vom Vater, sowie eine von der Mutter. Unterschiedliche Variationen des *Gens* werden als *Allel* bezeichnet. Wenn beide *Allele* des Individuums für ein bestimmtes Merkmal gleich sind, ist das Erbgut, bezogen auf dieses Merkmal, reinerbig oder homozygot. Liegen dagegen zwei verschiedene *Allele* vor, wird dies als mischerbig oder heterozygot bezeichnet. *Homozygotie* und *Heterozygotie* stellen den erwarteten Anteil an homozygoten bzw. heterozygoten *Genotypen* (Erbbild) in einer *Population* dar. Je höher die *Heterozygotie* desto höher ist die *Alleldiversität* und desto unwahrscheinlicher ist es, dass zwei zufällig ausgewählte Personen der *Population* denselben *Genotyp* aufweisen. Abbildung 2-5 zeigt diese zwei Möglichkeiten der Vererbung.

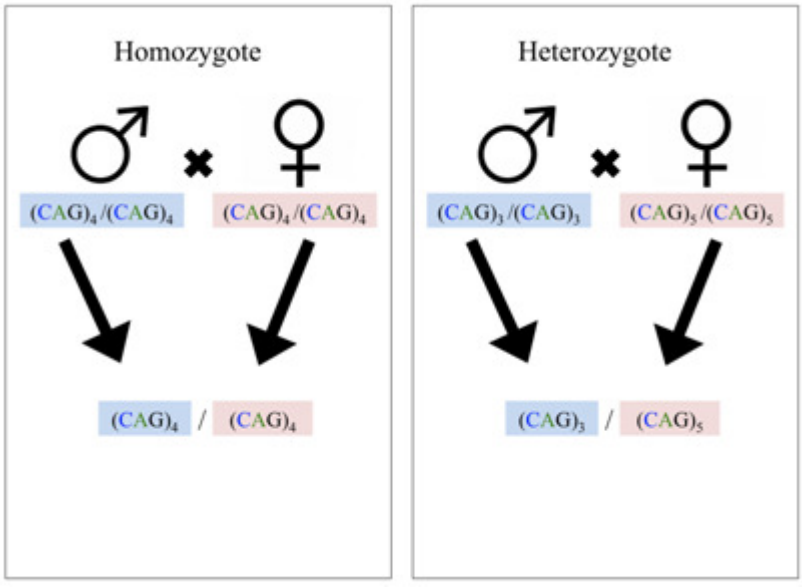


Abbildung 2-5: Homozygotie/Heterozygotie – (links) homozygote Vererbung, Mutter und Vater vererben für ein bestimmtes Merkmal das gleiche Gen $(CAG)_4$, (rechts) heterozygote Vererbung, Mutter und Vater vererben unterschiedliche Gene $(CAG)_3$ und $(CAG)_5$ [13]

Paternity Index (PI)

Der *Paternity Index* berechnet sich für einen untersuchten *Marker* aus Formel 2-1 und taucht oft bei der Auswertung von Vaterschaftstests auf.

$$PI = \frac{X}{Y}$$

Formel 2-1: Berechnung des Paternity Index - X=Wahrscheinlichkeit der Nullhypothese, Y=Wahrscheinlichkeit der Gegenhypothese [14]

X ist dabei die Wahrscheinlichkeit der Hypothese (Nullhypothese), dass der *Putativelternteil* tatsächlich der biologische Elternteil eines Kindes ist. Dem gegenüber steht die Wahrscheinlichkeit der Hypothese Y (Gegenhypothese). Diese Hypothese besagt, dass ein anderer Vater oder andere Mutter mit vergleichbarem ethnischem Hintergrund der gesuchte biologische Elternteil des Kindes ist. Der *Paternity Index* berechnet sich aus der Division der Wahrscheinlichkeiten der beiden Hypothesen und gibt bei der Auswertung von Populationsstudien an wie viel wahrscheinlicher es ist, dass der *Genotyp* eines Kindes die Hypothese X stützt, als das er die Gegenhypothese Y stützt. [15]

Power of Discrimination (PD)

Power of Discrimination bezeichnet die Wahrscheinlichkeit, dass zwei zufällig ausgewählte, nicht verwandte Personen verschiedene *Genotypen* aufweisen. Unverwandt müssen die Individuen deshalb sein, weil bei Blutverwandten eine größere Wahrscheinlichkeit für das Vorkommen derselben Merkmale besteht und infolgedessen durch Verwandte die Häufigkeitsverteilung verfälscht wird.

$$PD = 1 - 2\left(\sum_{i=1}^{n} p_i^2\right)^2 - \sum_{i=1}^{n} p_i^4$$

Formel 2-2: Berechnung der Power of Discrimination - pi=Frequenz des i-ten Allels in der Population, n=Anzahl der untersuchten Personen [14]

Der *Power of Discrimination* ergibt sich aus Formel 2-2 und gibt an wie hoch die Wahrscheinlichkeit für zwei zufällig ausgewählte Individuen ist, die gleiche Allelkonstellation für einen *Marker* zu besitzen. Der Wert variiert zwischen 0 und 1. Je

kleiner der Wert ist, desto geringer die Wahrscheinlichkeit. Beträgt der Wert z.B. 0,1, so bedeutet dies, dass diese Allelkonstellation statistisch bei einer von zehn nicht verwandten Personen anzutreffen ist. Je höher der Wert des *Power of Discrimination* ist, desto besser sind zwei Personen voneinander unterscheidbar. [16]

Mean Exclusion Chance (MEC)

Die allgemeine Vaterschaftsausschlusschance (engl. *Mean Exclusion Chance*, kurz *MEC*) ist von Bedeutung wenn sich die Frage stellt: „Mit welcher Wahrscheinlichkeit ein zu Unrecht der Vaterschaft beschuldigter Mann aufgrund der Untersuchung eines oder mehrerer *Marker* von der Vaterschaft ausgeschlossen werden kann." Aus Tabelle 2-1 ergibt sich z.B. für den *Marker* D3S1358 eine *MEC* von 0,5907. Das bedeutet, dass bei Untersuchung nur dieses einen *Markers* 59,07 % der zu Unrecht beschuldigten Männer der Stichprobe ausgeschlossen werden können. [10]

	STR-System	Chromosom	AVACH	Stichprobenumfang
1	TPOX	2	0.3887	2876
2	D3S1358	3	0.5907	2808
3	FGA	4	0.7232	9552
4	CSF1PO	5	0.4942	1643
5	D5S818	5	0.4851	1016
6	SE33	6	0.8971	6230
7	D7S820	7	0.6189	798
8	D8S1179	8	0.6331	500
9	TH01	11	0.5765	7373
10	VWA	12	0.6200	13667

Tabelle 2-1: Mean Exclusion Chance - chromosomale Lokalisierung, AVACH (Mean Exclusion Chance, kurz MEC) sowie die derzeit für Deutschland verfügbaren Stichprobenumfänge einzelner STR-Systeme (Marker) [10]

59,07 % ist natürlich ein geringes Maß für die Ausschlusswahrscheinlichkeit der Vaterschaft. In der Praxis werden mehrere Vaterschaftsausschlusswahrscheinlichkeiten der einzelnen *Marker* kombiniert. In der in Tabelle 2-1 gezeigten zehn *Marker* ergibt sich somit eine Vaterschaftsausschlusswahrscheinlichkeit von 99,9999 %. [10]

Es gibt verschiedene Formeln zur Berechnung der *MEC*. Welche verwendet werden können, hängt vom Markertyp (autosomal, X oder Y) ab. Die *MEC* für autosomale *Marker* wird nach der Formel 2-3 berechnet. [14]

$$MEC_{Krüger} = \sum_{i=1}^{n} p_i^3 (1-p_i)^2 + \sum_{i=1}^{n} p_i (1-p_i)^3 + \sum_{i<j}^{n} p_i p_j (p_i + p_j)(1-p_i-p_j)^2$$

Formel 2-3: Berechnung der Mean Exclusion Chance – p_i=Frequenz des i-ten Allels in der Population, p_j=Frequenz des j-ten Allels in der Population, n=Anzahl der untersuchten Personen [14]

2.2 Stand der Technik

Als Ziel der Diplomarbeit soll ein Prototyp entstehen, welcher publizierte Daten (*Allelfrequenzen*), zu möglichst vielen *Populationen* in einer Datenbank speichert und für forensische Wissenschaftler zur Verfügung stellt. Während der Analysephase (zu Beginn der Wissenschaftlichen Arbeit) wurden bereits vorhandene autosomale Datenbanken im nationalen sowie internationalen Umfeld nach den folgenden Kriterien untersucht und analysiert.

2.2.1 Analysekriterien

Die nachfolgend aufgezählten Kriterien dienen als Grundlage für die Analyse der ausgewählten Datenbanken und wurden mit Hilfe des Funktionsspezifikationskataloges für die autosomale Datenbank erarbeitet. [15]

Navigation

Dieses Kriterium gibt Auskunft darüber wie die verschiedenen Inhaltsseiten der Webanwendung zu erreichen und untereinander verknüpft sind. Der Nutzer sollte zu jeder Zeit einen Überblick darüber bekommen wo genau auf der Website er sich befindet, von wo er herkommt und wo er hin will. Alle Informationsseiten der Webanwendung sollten komfortabel durch ein Navigationsmenü erreichbar sein.

Usability

Usability oder auch Benutzerfreundlichkeit bezeichnet die vom Nutzer erlebte Nutzungsqualität bei der Interaktion mit dem System. Eine besonders einfache, zum Nutzer und seinen Aufgaben passende Bedienung wird dabei als benutzerfreundlich angesehen. Wird z.B. eine Hilfe angeboten oder wird der Nutzer durch Suchmasken unterstützt, um so schnell an die gewünschten Daten zu gelangen, ist dies ein positives Maß für die Usability.

Qualitätskontrolle

Um die Qualitätsanforderungen der eingepflegten Daten gewährleisten und nachvollziehen zu können ist es nötig dem Nutzer Referenzinformationen zu den einzelnen Datensätzen sowie *Markern* aufzuzeigen. Ebenso sollten die eingepflegten Daten einer strengen manuellen oder durch Werkzeuge unterstützten Qualitätskontrolle unterliegen.

Datensätze

Dieser Punkt stellt das Herzstück einer jeden autosomalen Datenbank dar, die eigentlichen *Allelfrequenzen* zu den *Populationen*. Der Nutzer sollte verschiedene Möglichkeiten geboten bekommen, schnell auf diese Daten zugreifen zu können. Es bietet sich an, die einzelnen Datensätze der Datenbank mit zusätzlichen Informationen auszustatten.

Populationsinformation

Details und Informationen zu den einzelnen *Populationen* verschaffen dem Nutzer zusätzlich Überblick und steigern somit den Komfort bei der Arbeit mit der Datenbank. Die Daten müssen vor dem Datenimport evaluiert werden und zusätzlich zu den *Allelfrequenzen* in die Datenbank gespeist werden.

Markerinformation

Wie die Populationsinformationen sollten auch die Inhaltsseiten der *Marker* mit hilfreichen Informationen erweitert werden um dem Anwender so schnell einen Überblick zu verschaffen auf welchem *Chromosom* sich ein *Marker* befindet sowie seine genetische und physikalische Lokalisierung. Der Einsatz von Synonymen zu den *Markern* ist vorteilhaft, da sich eine Vereinheitlichung der Nomenklatur noch nicht

durchgesetzt hat und somit verschiedene Begriffe für einen einzelnen *Marker* existieren. Literaturstellen zu den *Markern* sollen dieses Kriterium abrunden.

Mehrsprachigkeit

Um sich international etablieren zu können sollte eine Webanwendung in mehreren Sprachen angeboten werden und dem Nutzer in diesem Bereich keine Barrieren stellen. Prinzipiell sollte Englisch als Standardsprache und Deutsch als nationale Komponente speziell für deutschsprachige Kunden der Firma Qualitype AG angeboten werden.

Visuelle Features

Um die Anzeige einer Webanwendung visuell zu erweitern bietet es sich an die verschiedenen *Populationen* auf einer Weltkarte grafisch zu markieren. Der Nutzer bekommt so einen schnellen Überblick, zu welchen Regionen Populationsdaten in der Datenbank erfasst wurden bzw. einen Eindruck darüber vermittelt, wie stark die Regionen der Erde im Verhältnis zueinander mit Populationsdaten abgedeckt sind. Auch der Einsatz von Tabellen und Diagrammen steigert das visuelle Erlebnis einer Webanwendung erheblich.

biostatistische Berechnungen

Verschiedene Zusatzfunktionalitäten wie z.B. biostatistische Berechnungen (Kapitel 2.1.2) erweitern die Funktionspalette einer autosomalen Datenbank. Die berechneten Parameter treffen eine Aussage über die Qualität der Daten. Die Implementierung dieser Berechnungen wird vom Anwender als besonders hilfreich erachtet.

Aktualität

Unter Aktualität versteht man die kontinuierliche Pflege eines Systems sowie die ständige Einpflegung von Datensätzen.

2.2.2 Analyse bestehender Datenbanken

Im folgenden Abschnitt werden drei frei gewählte autosomale Datenbanken untersucht und analysiert und durch die oben beschriebenen Kriterien verglichen und bewertet. Während der Analysephase wurde auch eine in den Anlagen Teil 1 befindliche Vergleichstabelle erstellt. Die Analyse der bestehenden Datenbanken wird mit einem Fazit abgeschlossen.

The Distribution of the Human DNA-PCR Polymorphisms
(„Huckenbeck-Datenbank“)

Das erste der untersuchten Systeme ist die „Huckenbeck-Datenbank“ der Universität Düsseldorf. Diese Datenbank wurde von Wolfgang Huckenbeck und Hans Georg-Scheil ins Leben gerufen. Huckenbeck ist ein Rechtsmediziner an der Universität Düsseldorf. Das Projekt genießt ein hohes Ansehen im deutschen Forensik-Umfeld. Allerdings ist diese Datenbank nicht kontinuierlich gepflegt worden. Die letzte Aktualisierung der *Allelfrequenzen* liegt bereits drei Jahre zurück (Stand 2010). [16] Die Navigation der Webanwendung ist durch den geringen Inhalt einfach gehalten. Alle erfassten *Marker* werden auf der Hauptseite der Website nach Aktualisierungsdatum zur Anzeige gebracht. Von dort gelangt der Nutzer schnell zu den einzelnen Datensätzen. Die einzige Möglichkeit auf die Frequenztabellen zuzugreifen bietet sich über die *Marker* an. Die verschiedenen Populationstabellen mit den Frequenzwerten (*Allelfrequenzen*) sind nach Kontinenten angeordnet. Am Ende der Seite befinden sich Referenzen zu den jeweiligen Datensätzen. Somit sind die Quellen der Datensätze nachvollziehbar. Die Frequenztabellen sind fortlaufend angeordnet damit besteht das Problem, dass der Nutzer zum gesuchten Datensatz scrollen muss. Zusätzliche Informationen zu den Datensätzen, *Populationen* und *Markern* sind nicht angegeben. Die „Huckenbeck-Datenbank“ ist einfach gehalten und bietet so lediglich Auskünfte zu den einzelnen *Allelfrequenzen* der untersuchten *Populationen*. Zusätzliche Features wie biostatistische Berechnungen sucht man vergebens. Die Datenbank wird einsprachig in Englisch angeboten.

Autosomal STR DNA Database („STRDNA-Datenbank“)

Diese Datenbank befindet sich noch im Aufbau und ist daher inhaltlich noch wenig aussagekräftig (Stand 2010). Die Navigation der Website wird durch ein Menü im

Kopfteil der Seite realisiert. Von hier aus erreicht man alle Informationsseiten schnell und komfortabel. Bei der Suche nach den Datensätzen wird der Nutzer durch eine Weltkarte unterstützt. Durch einen Klick auf die gewünschte Region bekommt man alle *Populationen*, zu denen Datensätze gespeichert sind, angezeigt. Diese lassen sich dann als Microsoft Excel-Datei oder *Arlequin-Datei* exportieren. [17] Die Autoren der Seite verlassen sich auf die Aufnahmekriterien der jeweiligen forensischen Zeitschrift. Dieser Aspekt hat zur Auswirkung, dass die Daten keiner strengen Qualitätskontrolle unterliegen. [16]

Zusätzliche Informationen zu *Markern* oder *Populationen* sind bei der „STRDNA-Datenbank“ nicht aufgeführt. So bekommt man auch hier lediglich die „reinen“ *Allelfrequenzen* zu den *Populationen* angeboten.

Auf der Website wird erläutert, dass die Entwickler der Datenbank in Zukunft ein Tool zur Ähnlichkeitsberechnung planen. Damit ist es möglich zwei *Populationen* zu vergleichen, das heißt ihre Ähnlichkeit festzustellen. Bei einem genaueren Blick auf die Seite bekommt man jedoch eher den Eindruck, dass die Arbeiten am System eingestellt seien. Auch diese Datenbank wird einsprachig in Englisch angeboten.

The Allele Frequency Database („Alfred-Datenbank“)

Die letzte der untersuchten Datenbanken ist die „Alfred-Datenbank“. Sie ist zugleich die komplexeste hinsichtlich Funktionalität und umfasst dabei über 28.520.225 ständig wachsenden Frequenztabellen (Stand Oktober 2010). [18] Sie wird von der Yale University School of Medicine in New Haven betreut. Diese Datenbank speist sich aus vielen Quellen und vermittelt durch zahlreiche Links den Zugang zu weiteren externen Datenbanken. Sie zeichnet sich insbesondere dadurch aus, dass sie kontinuierlich gepflegt wird, so dass es sich mittlerweile um eine sehr umfangreiche Datenbank handelt. Ähnlich wie bei der „STRDNA-Datenbank“ besteht auch hier der Nachteil, dass die Daten keiner strengen Qualitätskontrolle unterliegen, was den Nutzer dazu zwingt, die Daten selbst zu überprüfen. [17] Ein weiterer Nachteil ist die zum Teil komplizierte Bedienung. Man erreicht zwar durch ein Navigationsmenü im Kopfbereich der Seite alle Unterseiten, wird jedoch durch die Fülle des Informationsangebotes überfordert. Es gibt zusätzliche Seiten zu *Populationen* und *Markern*, wo allgemeine Informationen zu diesen erläutert werden. Die eigentlichen Datensätze (*Allelfrequenzen*)

können über verschiedene Wege visuell zur Anzeige gebracht werden, beispielsweise über Diagramme oder Tabellen. Der Nutzer hat außerdem die Möglichkeit die Datenbank nach verschiedenen Parametern zu durchsuchen. Durch den Einsatz der Technologie *Extensible Markup Language* (*XML*) kann eine Suchanfrage jedoch durch die Komplexität der Datenmenge eine höhere Zeit in Anspruch nehmen. Die Datenbank wird einsprachig in Englisch angeboten.

2.2.3 Fazit

Abschließend zur Analyse der vorhandenen Datenbanken kann man sagen, dass die „Alfred- Datenbank“ heraus sticht. In der Fülle ihrer Funktionalität und Umfang an Inhalt stellt sie ein wirksames Werkzeug für die forensische Gemeinschaft dar. Die beiden anderen untersuchten Datenbanken konnten hinsichtlich der Kriterien (siehe Tabelle 2-2) nicht überzeugen. Die Resultate der Analyse dienen als Grundlage für die Anforderungsanalyse der autosomalen Datenbank.

	The Allele Frequency Database	Autosomal STR DNA Database	"Huckenbeck" Database
Navigation	++	-	o
Usability	o	+	+
Qualitätskontrolle	o	-	-
Datensätze	++	+	--
Populationsinformationen	o	o	o
Markerinformationen	++	--	-
Mehrsprachigkeit	++	--	-
Visuelle Features	+	o	o
biostatistische Berechnungen	--	--	--
Aktualität	--	--	--

Tabelle 2-2: Bewertung der Datenbanken – Kriterien (links), Datenbanken (oben), Kriterium erfüllt: ++ sehr gut, + gut, o durchschnittlich, - schlecht, -- sehr schlecht

2.3 Anforderungen an die autosomale Datenbank

Um eine für das forensische Umfeld optimal abgestimmte Datenbank zu entwickeln, wurden die drei untersuchten Datenbanksysteme im vorherigen Kapitel auf ihren Grundaufbau und Funktionalität untersucht. Diese Resultate, vom Autor der Arbeit eingebrachte Konzepte, sowie Spezifikationen aus dem Funktionsspezifikationskatalog sollen die Grundlage für ein grobes Anforderungsschreiben und später eine konkrete Anforderungsliste sein. [16]

Die Datenbank sollte in erster Linie grundlegende Informationen beinhalten. In diesen soll die Zielgruppe (forensische Wissenschaftler) der Datenbank klar definiert werden. Des Weiteren soll ersichtlich sein welche Einrichtung (Qualitype AG) die Datenbank zur Verfügung stellt. Der Nutzer soll schnell einen Überblick über die bereitgestellten Datensätze, deren Qualität und somit Funktionalitäten bekommen, sowie bei welchen Aufgaben ihn die Datenbank Unterstützung anbietet.

Durch ein gut sichtbares Navigationsmenü soll der Anwender schnell und komfortabel zu allen Unterseiten der Webanwendung navigieren können. Zusätzlich soll angeboten werden, die Daten auf direktem Wege über benutzerspezifische Suchmasken aus der Datenbank zu erhalten und somit direkt zum gesuchten Datensatz zu springen. Um eine optimale Orientierung im System zu gewährleisten sollte dem Nutzer zu jeder Zeit erkenntlich gemacht werden: „Wo im System befinde ich mich gerade?“, „Wo komme ich her?“ und „Wo möchte ich hin?“.

Für die Anzeige der eigentlichen Daten wie *Markern*, *Populationen* und *Allelfrequenzen* bieten sich dem Entwickler viele Möglichkeiten an. Es sollten jedoch große Tabellen und Textpassagen vermieden werden, sodass man die Seite immer als Einheit betrachtet. Vorteilhaft ist es, wenn Unterseiten mit Informationen zu *Markern* oder *Populationen* mit zusätzlichen Details ausgestattet sind und nicht „trocken“ präsentiert werden. Frequenztabellen mit den dazugehörigen Referenzen sollten dem Benutzer durch *Pagination* benutzerfreundlich zur Anzeige gebracht werden und ebenfalls mit Zusatzinformationen ausgestattet sein. Unzählig große Tabellen mit vielen Einträgen sollten nach verschiedenen Parametern gefiltert werden können. So ist es z.B. denkbar eine fiktive Tabelle Bevölkerungstabelle, welche verschiedene *Populationen* der Erde

beinhaltet, zur besseren Anzeige nach Kontinenten sortieren zu lassen. Der Benutzer hat es somit wesentlich einfacher den gewünschten Eintrag zu finden, da dieser in einer systematischen Grundordnung aufgelistet wird. Ein weiterer wichtiger Aspekt für die autosomale Datenbank, ist die kontinuierliche Einpflegung und damit in Verbindung stehend die Aktualität der Daten, da sich Forschungsmethoden ständig verbessern und weiterentwickeln. Der Anwender sollte immer den Eindruck haben sich auf einem stets aktuellen und gepflegten System zu befinden, sonst passiert es schnell, dass eine Webanwendung nicht mehr benutzt wird und somit in Vergessenheit gerät.

Nutzer auf der ganzen Welt sollen Daten übermitteln und hochladen können. Dies soll ihnen durch komfortable Eingabemasken erleichtert werden. Um diese Daten auf Echtheit und Qualität überprüfen zu können sollten sie jedoch im Vorfeld validiert werden und nicht auf direktem Wege, sondern über einen internen Kontrollmechanismus in die Datenbank gelangen. Das könnte z.B. dadurch realisiert werden, das ein fachkundiger Mitarbeiter der Firma Qualitype AG die Daten überprüft und nach erfolgreicher Eingangskontrolle ohne große Aufwendungen in die Datenbank speist.

Eine optionale Funktionalität ist die erfassten Daten zu den einzelnen *Populationen* über eine Weltkarte zu visualisieren, um so dem Nutzer schnell aufzuzeigen für welche *Populationen* der Erde Daten erfasst wurden. Während der Analyse der bestehenden Datenbanksysteme hat sich diese Funktionalität als sehr nützlich im Bezug auf die Navigation und Übersichtlichkeit der Seite erwiesen. Tabellen und Diagramme bieten sich hier ebenfalls als hilfreich an. *Idiogramme* zu den autosomalen *Chromosomen* können die verschiedenen *Marker* den jeweiligen *Chromosomen* zuordnen und dem Benutzer grafisch zur Anzeige gebracht werden. Es gibt die Möglichkeit die *Idiogramme* statisch oder dynamisch anzuzeigen. Dies macht für den Benutzer jedoch keinen großen Unterschied und stellt nur ein visuelles Feature dar.

Für die Bewertung von Populationsstudien sollen des Weiteren biostatistische Berechnungen für die Aussage zur Qualität der Daten durchgeführt werden können. Die berechneten Werte umfassen dabei u.a. die im Kapitel 2.1.2 aufgeführten biostatistischen Berechnungsparameter. Der Umfang dieser Berechnungsparameter soll sich an *GenoProof®2* orientieren. [9] Weitere Funktionalitäten wie die Berechnung von

Identitätswahrscheinlichkeiten und Ähnlichkeitsberechnungen können die Funktionspalette der Datenbank zu jedem Zeitpunkt erweitern.

Weitere Anforderungen sind dem Benutzer ein ausführliches und leicht verständliches Hilfesystem zur Verfügung zu stellen. Dies steigert die Usability des Systems. Ebenso kann die Usability durch ein Glossar, zum Nachschlagen von Fachbegriffen oder eine Bibliographie zur Recherche der Literaturstellen maßgeblich verbessert werden. Eine Sprachauswahl in den Standardsprachen Englisch und Deutsch, als nationale Sprachkomponente, sollen das Anforderungsschreiben abrunden.

3 Grundlagen der Informatik

In diesem Kapitel werden die theoretischen Grundlagen vermittelt, die mit der Diplomarbeit und hier speziell mit der Entwicklung des Prototyps von informationstechnischer Seite, in Verbindung stehen. Zunächst wird auf das *Model-View-Controller Modell* (*MVC-Modell*) und Webanwendungen im allgemeinen Sinne eingegangen und deren Vor- und Nachteile erläutert. Danach folgen allgemeine Angaben zur Java-Sprachspezifikation *Java Enterprise Edition* (*Java EE*), sowie der darin enthaltenen Kern-Technologie *Enterprise JavaBeans* (*EJB*). Abschließend wird über *Asynchronous JavaScript And XML* (*AJAX*), ein Programmiermodell für *Rich Internet Applications* (*RIAs*), berichtet. [19]

3.1 Model-View-Controller Modell

Im Laufe der Zeit hat sich für die Entwicklung von Webapplikationen ein Modell durchgesetzt, dass *Model-View-Controller Modell* siehe Abbildung 3-1 (ab jetzt *MVC-Modell*). Es wurde in den 70iger Jahren mit der ersten objektorientierten Programmiersprache Smalltalk-80 eingeführt. [20]

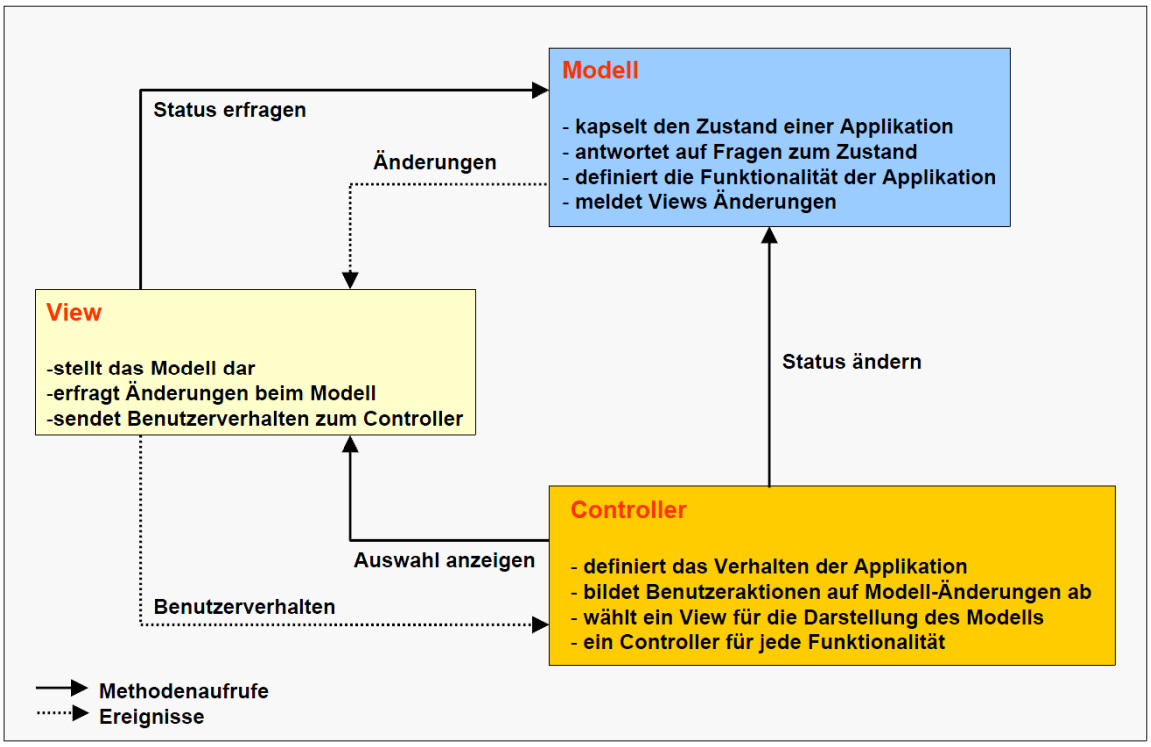


Abbildung 3-1: Model-View-Controller Modell – Erklärung siehe Abbildung [21]

Ziel des *MVC-Modells* ist es eine Anwendung wie in Abbildung 3-1 ersichtlich in drei Komponenten zu zerlegen. Die erste Komponente das „Modell“ dient zur internen Verarbeitung. Die zweite, die „View“, ist für die Ausgabe der Daten verantwortlich und die dritte, der „Controller“, für die Eingabe der Daten. Diese drei Komponenten werden separat behandelt und implementiert. Somit können Anwendungen leichter erstellt und deren Komponenten im Nachhinein leichter ausgetauscht werden. Dies dient der Wiederverwendung von Komponenten einer Anwendung. Außerdem ist es somit möglich, Fehler besser zu korrigieren, da diese besser lokalisiert werden können. Das *MVC-Modell* garantiert, dass Änderungen im Modell zu den entsprechenden Änderungen in der View führen. Im folgenden Abschnitt werden diese drei Komponenten näher beleuchtet. [22]

- **Modell**

  Das Modell repräsentiert die eigentlichen Daten und die Geschäftslogik einer Anwendung. Es wird oft ein Sachverhalt aus der realen Welt abgebildet. Neben der Verwaltung der Daten führt es auch alle Änderungen auf diesen aus. Das Modell liefert auf Anfrage den Zustand der Daten und reagiert auf Befehle diese zu ändern. In Java wird das Modell häufig über *JavaBeans* realisiert. *Java Enterprise Edition* setzt auf *Enterprise JavaBeans* (*EJB*, später in diesem Kapitel behandelt) als bevorzugte Modell-Komponenten-Technologie. Im Modell erfolgt oft ein Zugriff auf einen persistenten Speicher (z.B. relationale Datenbank). Das Modell stellt der Präsentationsschicht (View) seine Daten zur Verfügung. Dies geschieht durch entsprechende Methodenaufrufe im Modell. Weiterhin benachrichtigt das Modell die View über erfolgte Änderungen am Status, sodass die View immer aktuelle Daten anzeigt. [22]

- **View**

  Die View übernimmt die grafische Darstellung der Daten und ist für die visuelle Darstellung des Modells verantwortlich. Meist gibt es in einer Anwendung mehrere Views. Wenn sich Daten im Modell verändern, erstellt die View automatisch eine neue passende Darstellung des Modells und stellt diese grafisch dar. Jedes View-Objekt sollte die Möglichkeit besitzen ein Selbstupdate durchführen zu können, umso das Modell immer korrekt darzustellen. [21]

- **Controller**

  Der Controller definiert wie der Benutzer mit der Applikation interagiert. Er nimmt Eingaben vom Benutzer entgegen und bildet diese auf Funktionen des Modells oder der View ab. Somit benötigt er immer einen direkten Zugriff auf View und Modell. [21]

Vorteile

Durch die klare Trennung des Datenmodells von Präsentation und Steuerung können verschiedene Benutzeroberflächen oder Datendarstellungen benutzt werden ohne die Anwendung anpassen zu müssen. Des Weiteren kann eine Veränderung der Anzeigekomponenten völlig unabhängig von der Anwendungslogik stattfinden. Bei der Softwareentwicklung im Team wird durch die Entkopplung der Schichten eine leichtere Teilung der Aufgabengebiete zwischen Grafikdesigner, Programmierer und Textredakteur ermöglicht. [23]

Nachteile

Gegenüber den Vorteilen stehen auch einige Nachteile des *MVC*-Konzeptes, z.B. wird die hohe Komplexität, vor allem in kleinen Anwendungen, oft als unnötig angesehen. View und Controller werden im Modell zwar einzeln betrachtet, besitzen aber oft eine enge Beziehung. Dieser Fakt erschwert die klare Trennung von View- und Controller-Objekten während der Entwicklungsphase. Abhängig von der Schnittstelle zwischen Modell und View sind für die Darstellung der Daten in der View mehrere Aufrufe des Modells nötig. Dies kann die Geschwindigkeit der Anwendung verringern. [23]

3.2 Webanwendungen

Unter einer Webanwendung versteht man ein Computerprogramm welches auf einem Web-Server ausgeführt wird. Die Interaktion mit dem Benutzer erfolgt in der Regel über einen Web-Browser (z.B. Mozilla Firefox, Internet Explorer, Opera). [22] Der Benutzer (Client) und der Dienstanbieter (Server) sind über ein Netzwerk (z.B. Internet, Intranet) miteinander verbunden. Dadurch entsteht eine räumliche Trennung zwischen Client und Server. [24] Der Benutzer startet eine Webanwendung üblicherweise über die Eingabe des *Uniform Resource Locator* (*URL*) des Web-Servers [22] im Web-

Browser und der damit verbundenen Übermittlung einer Anfrage an den Server (*HTTP-Request*). Der Web-Server nimmt diese Anfrage entgegen und leitet diese an die Webanwendung weiter. Der Web-Server antwortet daraufhin z.B. in Form von *HTML*-Quellcodes (*HTTP-Response*). In Abbildung 3-2 ist ein solches Kommunikationsszenario zwischen Client und Server abgebildet.

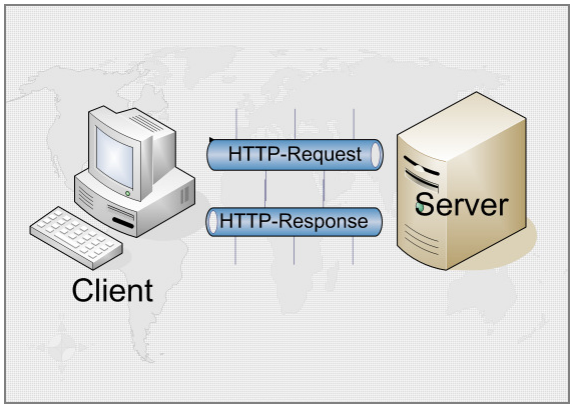


Abbildung 3-2: Client-Server-Kommunikation – Client (Benutzer) startet Anfrage an den Server (HTTP-Request), Server (Dienstanbieter) antwortet daraufhin (HTTP-Response)

Vorteile

Webanwendungen setzen auf der Clientseite nur einen Web-Browser voraus, dieser ist auf allen gängigen Betriebsplattformen (z.B. Windows, Linux) vorhanden, somit muss keine zusätzliche Software installiert werden. Durch diesen Fakt wird ein hohes Maß an Plattformunabhängigkeit erreicht, sofern bei der Entwicklung darauf geachtet wurde, dass alle Web-Browser unterstützt werden. Kommt es zu Änderungen der Programmlogik müssen diese nur zentral auf der Serverseite durchgeführt werden. Durch die immer stärkere Verbreitung von Web-Browsern auf anderen Endgeräten (z.B. Mobiltelefone, PDAs) finden Webanwendungen eine rasante Verbreitung jenseits der klassischen Softwareumgebung. [24]

Nachteile

Webanwendungen benötigen für die Kommunikation zwischen Client und Server eine ständige *TCP/IP*-Verbindung. [22] Eine dauerhafte Verbindung kann nur durch *Sessions* erreicht werden. Daraus ergeben sich meist Sicherheitsprobleme. Weiterhin ist zu bemängeln, dass die Datenrate der Verbindung auf die Webanwendung ausgelegt sein muss. Webanwendungen sollten im Normalfall auf allen Web-Browsern funktionieren, was nicht selbstverständlich ist da jeder Web-Browser die *Hyper Text Markup*

Language (*HTML*) unterschiedlich interpretiert. Dies führt meist zu Anzeigeproblemen auf Clientseite. [24]

3.3 Java Enterprise Edition

Die *Java Enterprise Edition* (von nun an *Java EE*) bezeichnet die Spezifikation einer Softwarearchitektur insbesondere für Webanwendungen, welche sich speziell für Unternehmensanwendungen durchgesetzt hat. In dieser Spezifikation werden Softwarekomponenten und Dienste definiert, die primär in der Programmiersprache Java erstellt werden. [25] Klare Schnittstellen zwischen diesen Komponenten und Schichten sorgen dafür, dass Softwarekomponenten unterschiedlicher Hersteller kompatibel sind solange sich diese an die Spezifikationen halten. Die *Java EE-APIs* erweitern die *Java Standard Edition-APIs* (*Java SE-APIs*) um zusätzliche Technologien (z.B. *Java ServerFaces*, *Java Persistence API*, *Enterprise JavaBeans*). [26] Diese Komponenten benötigen als Laufzeitumgebung eine besondere Infrastruktur, so dass ein spezieller *Applikationsserver* (z.B. *JBoss Application Server*, Apache Tomcat oder IBM WebSphere) verwendet werden muss. Standardmäßig basiert eine *Java EE*-Anwendung auf der *Drei-Tier-Architektur* siehe Abbildung 3.3. [22]

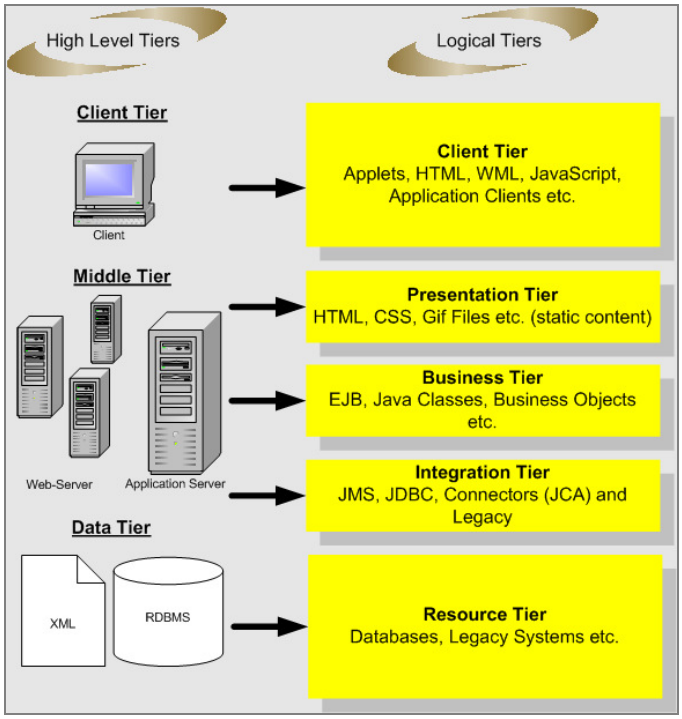


Abbildung 3-3: Java Enterprise Edition-Tiers – Java EE-Anwendung wird in drei Schichten zerlegt (links, High Level Tiers), Client Tier, Middle Tier und Data Tier, Technologien und Funktionen der dazugehörigen Schichten (rechts, Logical Tiers)

Wie zu sehen ist wird die Anwendung dabei grob in drei Schichten untergliedert. Die Präsentationsschicht (erste Schicht), wird auch als Client-Tier bezeichnet. Sie dient zur Präsentation der Daten im Web-Browser, nimmt Benutzer-Events entgegen und steuert das Benutzer-Interface. Den Kern der Software bildet die Applikationsschicht (zweite Schicht) oder auch Middle-Tier genannt. Sie enthält die gesamte Geschäftslogik und stellt für die Präsentationsschicht Dienste zur Verfügung. Des Weiteren schützt sie die Datenerhaltungsschicht (dritte Schicht), welche als Data-Tier bezeichnet wird. Diese ist für das Sichern und Anbieten von Daten verantwortlich und wird meist durch ein Datenbanksystem repräsentiert.

Vorteile

Die *Java EE*-Spezifikation stellt einen standardisierten Rahmen für die Entwicklung komplexer Anwendungen zur Verfügung. *Java EE* erlaubt einen modularen Aufbau und bietet fest definierte Schnittstellen zwischen den Komponenten. Sie besticht dadurch, dass die Anwendungen leicht skalierbar und plattformunabhängig sind. Durch die *Drei-Tier-Architektur* wird eine klare Trennung der einzelnen Schichten ermöglicht, somit können die einzelnen Module einfach ausgetauscht und wieder verwendet werden. Außerdem steht durch die Popularität der Programmiersprache Java eine breite Entwicklergemeinde zur Verfügung. [27]

Nachteile

Ein erheblicher Nachteil ist der starke Ressourcenverbrauch der zum Einsatz kommenden *Applikationsserver*, auf dem die *Java EE*-Anwendungen laufen. Dieser wird durch die große Funktionalität begründet und muss durch den Einsatz teurer Hardwarekomponenten (z.B. Speicher und Prozessor) ausgeglichen werden. Dies führt zwangsläufig dazu, dass die Ausgaben für den *Applikationsserver* einer *Java EE*-Anwendung deutlich höher ausfallen. Ein weiterer Nachteil besteht darin, dass eine so komplexe Spezifikation einen großen Einarbeitungsaufwand in die jeweiligen Technologien erfordert. An die Mitarbeiter wird eine hohe Anforderung von Wissen und Erfahrung für die Erstellung komplexer Geschäftsanwendungen gestellt. [27]

3.4 Enterprise JavaBeans

Mit *Enterprise JavaBeans* (*EJB*) existiert in der *Java Enterprise Edition* (*Java EE*) eine serverseitige Komponententechnologie mit der sich verteilte und mehrschichtige Geschäftsanwendungen entwickeln lassen. [22] Mit ihrer Hilfe können wichtige Konzepte für Unternehmensanwendungen, wie beispielsweise automatisierte Namens-, Transaktions-, Persistenz-, Verteilungs- oder Sicherheitsfunktionen umgesetzt werden. Die Spezifizierung sieht vor, dass sie nur innerhalb eines *Applikationsservers* verwendet werden können. Dieser übernimmt betriebssystemnahe Managementaufgaben, wie *Thread-Pooling*-, Ressourcenverwaltung und die Lastenverteilung im Netz. [28] Der Zugriff auf die *EJBs* erfolgt dabei über einen *Container*, welcher serverseitige Aufgaben, wie das Lebenszyklusmanagement und die Synchronisation von Zugriffen regelt. Der *Container* ist im *Applikationsserver* eingebettet (siehe Abbildung 3-4). [22]

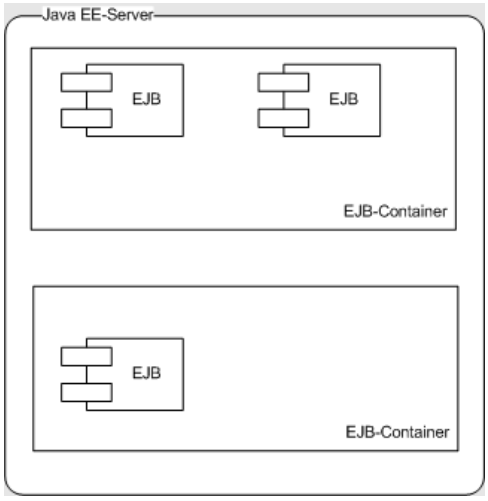


Abbildung 3-4: Enterprise JavaBeans-Architektur – Die EJBs befinden sich zur Laufzeit in Containern, der Container steuert den Zugriff auf die EJB, betriebssystemnahe Managementaufgaben werden vom Applikationsserver (hier Java EE-Server) vorgenommen

Es gibt drei Grundtypen von *EJBs* diese werden im nachfolgenden kurz erläutert:

- **Entity Beans**

  Die erste Säule des *EJB*-Konzeptes stellen die *Entity Beans* dar. Sie modellieren persistente Anwendungsobjekte in einer Webanwendung und besitzen Attribute, welche in einer Datenbank gespeichert werden können. Die Zugriffe auf diese Attribute erfolgen über *Getter*- und *Setter-Methoden*. In der autosomalen Datenbank wird z.B. der *Marker* durch eine *Entity Bean* repräsentiert. Er besitzt

spezifische Attribute wie z.B. einen Namen oder einen Größenbereich. Im Allgemeinen bildet die *Entity Bean* eine Tabelle und das instanzierte Objekt einen dazugehörigen Datensatz der Tabelle ab, dieser kann dabei von mehreren Clients parallel genutzt werden. Es gibt zwei rudimentäre Möglichkeiten *Entity Beans* zu persistieren. Zum einen die Verwaltung durch den *Container*. Dieser Vorgang wird als *Container Managed Persistence* (*CMP*) bezeichnet und stellt einen automatisierten Mechanismus dar. Es gibt aber auch die Möglichkeit, dass der Entwickler die Persistierung der Daten von Hand vornimmt, dieser Prozess wird *Bean Managed Persistence* (*BMP*) genannt. [22] In der autosomalen Datenbank wird die Persistierung der *Entity Beans* durch das Persistenzframework *Hibernate* geregelt (siehe Kapitel 5.2). *Hibernate* erzeugt dabei das Datenbankschema, legt die Tabellen an und bildet die *Entity Beans* in einer relationalen Datenbank ab. Somit wird dem Entwickler in diesem Bereich eine Vielzahl von Arbeit abgenommen. Die Definition einer *Entity Bean* wird durch die *EJB* 3.0-Spezifikation der *Java Persistence API* (*JPA*) vorgegeben. [28] Zusätzliche Annotationen sorgen dabei für eine leichte und übersichtliche Konfiguration.

- **Session Beans**

  Die zweite Säule bilden die *Session Beans*. Sie realisieren die Logik des Geschäftsprozesses. Beispielsweise wird das Reservieren eines Hotelzimmers über eine Webanwendung mittels einer *Session Bean* realisiert oder im Bereich der autosomalen Datenbank das Suchen von *Allelfrequenzen*. Sie sind für die Steuerung der Interaktion zwischen den *Entity Beans* zuständig. Im Unterschied zu den *Entity Beans* sind sie kurzlebig, nicht persistent und werden nicht in der Datenbank gespeichert, trotzdem sind mit ihnen Datenbankoperationen möglich. Sie werden in zwei Gruppen unterteilt, zustandslose und zustandsbehaftete. Beide decken unterschiedliche Aspekte der Anwendungsentwicklung ab. Bei den zustandslosen bleiben die Zustände der Instanzvariablen nicht erhalten und können dadurch effizient vom *EJB-Container* verwaltet werden. Dagegen wird bei den zustandsbehafteten *Session Beans* der *Conversational State* innerhalb einer Sitzung aufrechterhalten. Ein typischer Anwendungsfall ist die Modellierung eines Warenkorbes von webbasierten Verkaufsportalen, in der Produkte hineingelegt und wieder herausgenommen werden können. Der Zustand des

Warenkorbs bleibt während der gesamten Sitzung erhalten. Dieser Vorgang ist jedoch nicht mit der Persistierung von *Entity Beans* vergleichbar. [22]

- **Message Driven Beans**

  Die letzte Säule wird durch die *Message Driven Beans* (*MDB*) repräsentiert. Sie ergänzen die bereits in früheren Spezifikationen definierten *Entity Beans* und *Session Beans* um serverseitige Komponenten und können für den asynchronen Nachrichtenaustausch verwendet werden. Die Beans selbst besitzen keinen Zustand und stehen in einem Pool für die Verwendung bereit. [22]

3.5 *Asynchronous JavaScript And XML*

Asynchronous JavaScript And XML (ab jetzt *AJAX*) ist ein Ansatz zur Programmierung von *Rich Internet Applications* (*RIAs*). *AJAX* steht für folgende Idee: Inhalte sollen dynamisch von einem Server abgerufen und in die zugehörigen *UserInterface-Komponenten* (*UI-Komponenten*) einer Webseite eingefügt werden ohne die gesamte Seite neu zu laden. Durch *AJAX* entwickelte sich eine neue Generation in der Entwicklerszene. Verhaltensweisen wie *Drag&Drop* und andere Funktionalitäten waren bisher nur in Desktopanwendungen möglich. Mit *AJAX* kann man dieses Verhalten nun auch in eine Webanwendung integrieren. Dadurch steigert sich die Benutzerfreundlichkeit einer Anwendung und minimiert deren Ladezeit. Alte Webanwendungen simulierten ein Verhalten wie in Abbildung 3-5 zu sehen ist. [29]

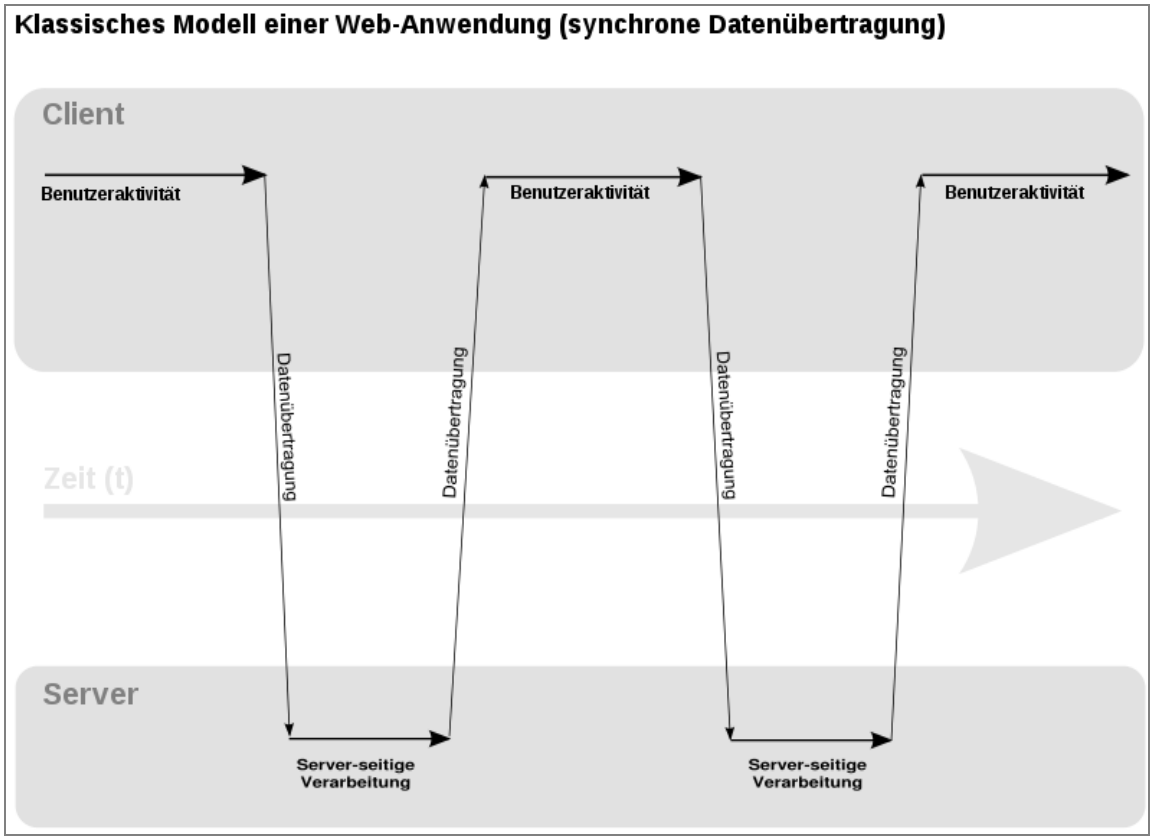


Abbildung 3-5: Klassisches Modell einer Web-Anwendung – synchrone Datenübertragung zwischen Client und Server, Benutzer muss auf die Antwort des Servers eine variable Zeit (t) warten um fortfahren zu können [30]

Wenn der Nutzer auf einen Button in der Anwendung klickte, musste er warten bis der Server die Anfrage (*HTTP-Request*) vollständig abgearbeitet und geantwortet (*HTTP-Response*) hatte. Durch *AJAX* wird nur noch der benötigte Inhalt vom Server geladen.

Die Anfrage wird clientseitig durchgeführt und läuft somit deutlich schneller ab, sodass der Nutzer auch während der Anfrage (*AJAX-Request*) mit dem System interagieren kann (siehe Abbildung 3-6). [29]

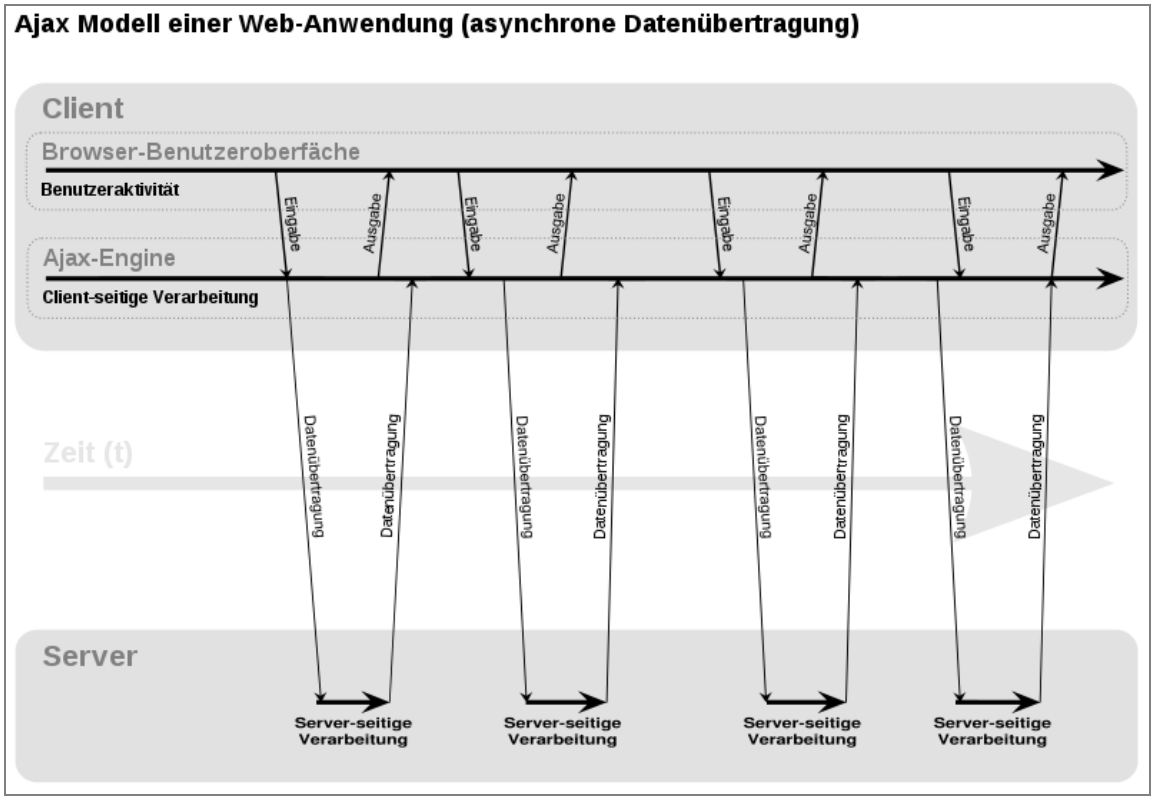


Abbildung 3-6: AJAX-Modell einer Web-Anwendung – asynchrone Kommunikation zwischen Client und Server, die Verarbeitung der Anfrage findet clientseitig durch die AJAX-Engine statt, der Nutzer kann auch während der Anfrage mit dem System interagieren, die Reaktionszeit (t) verkürzt sich [30]

4 Entwicklung von Webapplikationen mit Frameworks

Obwohl es möglich ist, auch mit rudimentären Technologien und Architekturen (z.B. *Java ServerPages*, *Hypertext Preprocessor* oder *Hypertext Markup Language*), Webanwendungen zu erstellen, werden viele Funktionalitäten von Webanwendungen kaum oder gar nicht abgedeckt. Dazu gehören u.a. Internationalisierung, Validierung, Verwaltung von Formulardaten und Modularisierung. Die einzelnen Technologien bieten zwar die Möglichkeit für eine solche Umsetzung an, sind jedoch nur mit großem programmiertechnischen Aufwand zu realisieren.
Um den Entwickler während des Entwicklungsprozesses zu entlasten, finden Web-Frameworks ihren Einsatz. Dies führt zu einer deutlichen Minimierung der Entwicklungszeit. In der Qualitype AG wird firmenweit auf die Java-Technologie gesetzt. Anhand der Vorteile von Frameworks und dem zu Grunde liegenden Know-hows rund um die Programmiersprache Java seitens der Qualitype AG soll der Prototyp mit Technologien aus dem Gebiet Java umgesetzt werden.
In diesem Kapitel werden Analysekriterien vom Autor definiert. Im Anschluss daran werden mit Hilfe dieser Kriterien zwei Web-Frameworks, aus dem Bereich Java (JBoss *Seam* und *Spring MVC*), analysiert. Anschließend wird das Framework JBoss *Seam* näher beleuchtet.

4.1 Evaluation von Frameworks

Für die Entwicklung komplexer Webanwendungen bieten sich dem Entwickler eine Vielzahl von Frameworks zur Unterstützung an. Während der Entscheidungsphase der Diplomarbeit wurden einige Frameworks hinsichtlich ihrer Funktionalität untersucht. Bei der Untersuchung kamen JBoss *Seam* und *Spring MVC* in die engere Wahl. In dem folgenden Kapitel werden diese zwei ausgewählten Web-Frameworks anhand der folgenden Kriterien untersucht und evaluiert.

4.1.1 Analysekriterien

Diese Kriterien wurden gemäß den Anforderungen an die autosomale Datenbank vom Autor erarbeitet und aus Entwicklersicht beschrieben. Die nachfolgenden Kriterien werden im Folgenden allgemein oder anhand der autosomalen Datenbank näher erläutert.

Installation

Entscheidet sich ein Entwickler für den Einsatz eines bestimmten Frameworks, wird er als erstes mit der Installation konfrontiert. Dieses Kriterium bestimmt den Zeitaufwand und die Einfachheit bei der Einrichtung der notwendigen Entwicklungsumgebung. Es ist also Vorteilhaft, wenn die Anbindung an eine *Integrated Development Environment* (*IDE*), wie beispielsweise *Eclipse* oder *NetBeans*, problemlos erfolgen kann.

Einarbeitungsaufwand

Dieser beschreibt den Aufwand eines Entwicklers, um sich mit dem Framework und dessen Funktionalitäten vertraut zu machen. Dieses Kriterium kann sicherlich nicht objektiv bewertet werden. Dennoch ist hierbei darauf zu achten, ob mit standardisierten Konzepten gearbeitet wurde, welche bereits aus anderen Technologien bekannt sind. Ebenso maßgeblich für die Einarbeitung in ein neues Gebiet ist eine vollständige Dokumentation. Dieses Kriterium wird im nachfolgenden Absatz separat erläutert.

Dokumentation

Die Qualität der Dokumentation eines Frameworks kann für dessen Verwendung ausschlaggebend sein. Bei der Auswahl eines Frameworks ist stets darauf zu achten, dass die Referenz-Dokumentation aktuell und vollständig ist. Außerdem ist es von Vorteil, wenn in diversen Foren bzw. Blogs Beispielanwendungen und Tutorials anderer Entwickler zur Verfügung stehen und somit ein großes Ökosystem rund um das Framework besteht.

Konfigurationsaufwand

Um die Vielzahl an Funktionalität nutzen zu können und an die speziellen Anforderungen anzupassen, bedarf es bei den meisten Frameworks einer Menge an Konfigurationseinstellungen. Allerdings stehen damit auch Probleme in Verbindung.

Die Einstellungen werden meistens in *Extensible Markup Language*-Dateien (*XML*-Dateien) vorgenommen. Dies führt zu einer unübersichtlichen Konfigurationsstruktur und kann nur mit viel Aufwand und Kenntnis gewartet werden.

Hibernate

Da die Rohdaten der autosomalen Datenbank persistiert werden müssen, muss das Framework eine geeignete Schnittstelle für eine relationale Datenbank zur Verfügung stellen.

Dieses Kriterium wurde ausgewählt, weil dem Entwickler in diesem Punkt speziell im Bereich der Konfiguration für die Datenanbindung viel Arbeit abgenommen werden sollte, da sonst ein zusätzlicher Arbeits- und Lernaufwand entstehen würde. Erweiterungen oder Änderungen der Datenbankstruktur sollten sich unproblematisch durchführen lassen. Des Weiteren sollte sich der Entwickler nicht auf eine Datenbanktechnologie beschränken müssen, sodass er im Bereich der Datenbankauswahl flexibel bleibt.

Navigation

Um das Verhalten einer komplexen Webanwendung optimal von Entwicklerseite administrieren zu können, sollte das Framework eine geeignete Methode für die Konfiguration der Navigation bieten. Speziell für die Übersichtlichkeit und Erweiterbarkeit ist dieses Kriterium ein wichtiger Bestandteil.

Internationalisierung

Da die autosomale Datenbank ein breites Spektrum an Wissenschaftlern und Forschern aus dem forensischen Umfeld ansprechen sollte, ist das Kriterium Internationalisierung unabdingbar. Internationalisierung bedeutet, dass die Seiten einer Webanwendung gemäß der Sprache und Kultur des Benutzers angezeigt werden. Jeder Web-Browser besitzt bestimmte Länder und Spracheinstellungen. Das Framework sollte in der Lage sein, diese Angaben aus dem *HTTP-Request* auszulesen und der Applikation zur Verfügung zu stellen. Die Spracheinstellungen sollten vom Java-Code getrennt sein und können so extern konfiguriert und automatisch zur Anzeige gebracht werden.

Authentifizierung und Autorisierung

Einige Funktionalitäten der autosomalen Datenbank sollen nur für ausgewählte Nutzer zugänglich sein. Frameworkspezifische Verfahren können dabei den Verwaltungsaufwand reduzieren und die Wartbarkeit sowie Übersichtlichkeit verbessern.

View-Technologie

Um einen hohen Grad an Benutzerfreundlichkeit zu bieten, sollte das Framework eine ausgereifte View-Technologie mit einer umfangreichen *Komponentenbibliothek* zur Verfügung stellen. Vorteilhaft ist es, wenn das Framework mehrere View-Technologien unterstützt. Ansprechende und barrierefreie Benutzeroberflächen sind in der heutigen Zeit ein Maß für die Anzahl der Benutzer für einen Webauftritt.

4.1.2 Analyse der Frameworks

Im folgenden Abschnitt werden die zwei ausgewählten Frameworks JBoss *Seam* und *Spring MVC* bezüglich der oben genannten Kriterien überprüft. Anfangs werden theoretische Vor- und Nachteile genannt. Aus den gesammelten Resultaten werden zum Schluss Folgerungen auf die Praxistauglichkeit im Bezug auf die autosomale Datenbank benannt. Den Vergleich der beiden Frameworks soll ein Fazit, wo die Entscheidung mit der entsprechenden Begründung wiedergegeben wird, abrunden.

JBoss Seam

Von seinen Entwicklern wird JBoss *Seam* (kurz als *Seam* bezeichnet) als leichtgewichtiges Framework für *Java EE* 5.0 bezeichnet. [31] Es vereint eine Vielzahl an Technologien (z.B. *Java ServerFaces*, *Enterprise JavaBeans* 3, *Asynchronous JavaScript and XML* oder *Java Business Process Management*) und stellt eine wesentliche Erleichterung für die Entwicklung zustandsbehafteter und auf Geschäftsprozesse bezogener Anwendungen dar. [32] *Seam* wurde ebenfalls, wie die beliebte *Object-Relational Mapping*-Lösung *Hibernate*, von Gaven King entwickelt. [31]

Seam Anwendungen bestehen aus sogenannten Komponenten. Die *Seam*-Komponenten werden von der *Seam*-Laufzeitumgebung verwaltet. Sie kümmert sich um den gesamten Lebenszyklus, wie beispielsweise die Instanzierung oder Zerstörung der einzelnen

Komponenten. Diesen Vorgang nennt man *Inversion of Control* (*IoC*), da der Entwickler die Kontrolle abgibt Instanzen von Klassen zu erzeugen. Aus diesem Grund sieht man den Operatur „new“ zur Instanzierung von speziellen Objekten in *Seam*-Anwendungen eher selten. Eine weitere Funktionalität ist die *Dependency Injection* (*DI*). Bei diesem Vorgang werden Instanzen von Klassenkomponenten in andere Komponenten injiziert. Dies bringt den Vorteil mit sich, dass der Entwickler lose gekoppelten Quelltext erhält und harte Abhängigkeiten zu konkreten Implementierungsklassen vermeidet. Im Bereich Konfiguration setzt *Seam* auf *Annotations* im Sourcecode. Diese ersetzen die unzähligen *XML*-Konfigurationsmöglichkeiten. [31]

Seam verbindet die in Abbildung 4-1 gezeigten Technologien und besitzt die Struktur des *MVC-Modells* (siehe Kapitel 3.1). Im Bereich View-Technologie wird in *Seam* auf *Java ServerFaces* (kurz *JSF*, später im Kapitel) gesetzt. *JSF* ist die Standard-Präsentations-Technologie in der *Java Enterprise Edition* (*Java EE*) und bietet ein großes Ökosystem von Anwendern und Anbietern. Ebenso stehen auch andere Möglichkeiten für die Anzeige (z.B. *Java ServerPages*) zur Verfügung. [33] Die Controller- und Modell-Schicht wird durch *Enterprise JavaBeans* (*EJB*, später in diesem Kapitel) repräsentiert. *Seam* bietet dabei die Möglichkeit direkt von der Präsentationsschicht auf *EJB*-Komponenten zuzugreifen.

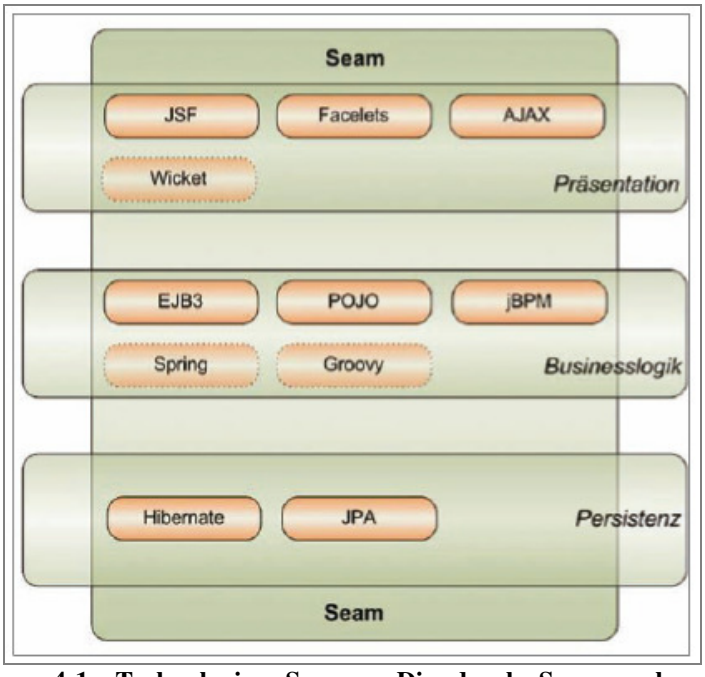


Abbildung 4-1: Technologien Seam – Die durch Seam verbundenen Technologien eingebettet in eine Drei-Tier-Architektur [34]

Des Weiteren bringt *Seam* einen Generator (*seam-gen*) mit. Mit ihm lassen sich in kurzer Zeit vorkonfigurierte Projekte erstellen. [31] Diese liefern eine fertige Projektstruktur mit allen benötigten Bibliotheken, Konfigurationsdateien, *JSF*-Seiten und einen *build-Script*. Der Generator *seam-gen* wird direkt mit *Seam* ausgeliefert und ist ein wirksames Werkzeug für die Erstellung von *CRUD-Anwendungen*. Durch seinen scriptbasierten Befehlsansatz findet der Generator in beliebigen Entwicklungsumgebungen Einsatz. Zudem sind diese Projekte standardmäßig mit den *IDE´s* wie *Eclipse* oder *Netbeans* kompatibel.

Die Navigation in *Seam*-Projekten kann klassisch über *JSF* erfolgen. Dabei werden durch die Auswertung von Rückgabewerten oder Actions interne Abläufe definiert. Diese Navigationsregeln werden standardmäßig in der navigation.xml-Datei administriert. Doch die *JSF*-Navigationsregeln selbst sind statisch und zustandslos. Darum bietet *Seam* auch in diesem Bereich Alternativen an. Da es von Haus aus mit zustandsbehafteten Komponenten arbeitet, können auch Zustandsobjekte in den Navigationsfluss eingebaut werden. Beispielsweise kann man die gesamten Informationen über den Seitenfluss einschließlich der Übergangsbedingungen für den jeweiligen Anwendungszustand in einem *JPDL*-Dokument zusammenfassen (*jBPM Process Definition Language* (*JPDL*) = *Java Business Process Management* (*jBPM*)). Das *JPDL*-Dokument ersetzt sowohl die statischen Navigationsregeln in der navigation.xml-Datei als auch die Rückgabewerte der *UserInterface-Eventhandler-Methoden*. [31] Für kleinere Anwendungen, wie die autosomale Datenbank ist *JPDL* zu komplex, weshalb der Autor nicht näher darauf eingehen wird.

Wie oben erwähnt, ist *Seam* aus der *Hibernate*-Community entstanden. Von daher wird *Hibernate* hervorragend implementiert. Der Generator *seam-gen* stellt dabei die *Hibernate*-Bibliotheken automatisch zur Verfügung.

Obwohl eine *Seam*-Anwendung auf *Java EE*-Technologien basiert, benötigt sie keinen speziellen Server, sodass selbst eine einfache *Servlet-Engine* wie z.B. Apache Tomcat verwendet werden kann. Arbeitet man jedoch mit *EJB* 3 wird ein spezieller *Applikationsserver*, wie beispielsweise der *JBoss Application Server* (siehe Kapitel 6.1.2), benötigt. [31]

Für den Bereich Authentifizierung und Autorisierung bietet *Seam* ebenfalls einfache aber zugleich sehr wirksame Werkzeuge an. Somit lassen sich beispielsweiße mit einer Codezeile eine komplette Login-Funktionalität implementieren. Das von *Seam* mitgelieferte Security-Framework JBoss *Rules* (früher *Drools*) basiert auf Regeln. In diesen kann der Entwickler festlegen welche Nutzer auf eine Seite, *UI-Komponente* oder *Bean*-Methode, zugreifen dürfen. Diese Sicherheitsregeln sind zustandsbehaftet. Das Security-Framework zeichnet sich durch seine Leistungsstärke und Flexibilität aus und kann in fast allen Webanwendungen verwendet werden. [33]

Seam stellt auch Möglichkeiten zur Internationalisierung bereit. Die verschiedenen länderspezifischen Angaben werden in zentral abgelegten *Properties-Dateien* vorgenommen. Auf diese kann dann je nach Web-Browser-Konfiguration oder Parameterübergabe zugegriffen werden. [31]

Spring MVC

Spring MVC ist wie *Seam* ein Web-Framework für die Entwicklung webbasierter Anwendungen und zählt zu den Teilprojekten des *Spring*-Frameworks. [35] *Spring* wurde von Rod Johnson unter der Leitung der Firma Spring Source entwickelt und ist zurzeit in der Version 3.0.3 (Stand 2010) erhältlich. Auch dieses Projekt entstand aus Frustration über die oft unnötige Komplexität bestehender *Java EE*-Architekturen und stellt somit eine praktikable Alternative zu diesen her. [36]

Spring MVC basiert auf den beiden Konzepten der *Dependency Injection* (siehe *Seam* Kapitel 4.1.2) sowie der *Aspektorientierten Programmierung* (*AOP*). In einem Programm kann es Aufgaben geben die an mehreren Stellen des Programmcodes vorkommen wie beispielsweiße Datenbank-Zugriffe, Caching oder Authentifizierung. Dieses mehrfache Auftreten des gleichen Codes an unterschiedlichen Stellen widerspricht dem *Don´t repeat yourself-Konzept* (*DRY-Konzept*). [37] Die *Aspektorientierte Programmierung* ermöglicht es, solche Aufgaben zu kapseln, so dass die Aufgabe nur einmal programmiert werden muss, aber an mehreren Stellen ausgeführt werden kann.

Für die Navigation in einer Webanwendung wird *Spring MVC* durch ein eigenes entwickeltes Framework namens *Spring Web Flow* unterstützt. Mit ihm lassen sich

Navigationsregeln erstellen sowie Zustände verwalten. Ein Web Flow beschreibt dabei eine funktional zusammengehörige Abfolge von Benutzerinteraktionen und hat damit oft einen direkten Bezug zu *Use Cases* aus der objektorientierten Analyse. Die *Use Cases* lassen sich dabei durch das Navigations-Framework *Spring Web Flow* beschreiben. Flows können wieder verwendet werden und stellen somit einen großen Vorteil des Navigations-Frameworks dar. [38]

Der große Konfigurationsaufwand in einer frameworkbasierten Webanwendung kann auch in *Spring MVC* mit Hilfe von *Annotations* administriert werden. Somit bleibt gerade unerfahrenen Entwicklern eine Menge an Konfigurationsaufwand per *XML* erspart.
Spring MVC unterstützt die Integration eines *Object-Relational Mappers* wie z.B. *Hibernate*. Der Entwickler erhält neben einer Vereinheitlichung der Konfiguration vor allem ein durchgängiges Programmiermodell. Die Datenbankanbindung ist dabei relativ lose an die Anwendung gekoppelt und kann somit später problemlos durch eine andere Integration ersetzt werden. Unter diesen Gesichtspunkten ist deutlich ersichtlich, dass sich *Hibernate* und *Spring MVC* gut ergänzen ohne dabei gegenseitige Abhängigkeiten aufzuzeigen. [36] In Abbildung 4-2 sind zwei Aspekte ersichtlich. Einerseits wie sich *Hibernate* und *Spring MVC* in eine Anwendungslandschaft integrieren lassen. Dies wird hier mit der *Drei-Tier-Architektur* aufgezeigt. [47] *Spring MVC* befindet sich dabei in der Middle-Tier. *Hibernate* findet seinen Platz dagegen im Persistenzbereich, d.h. bei der Anbindung zur Datenbank.

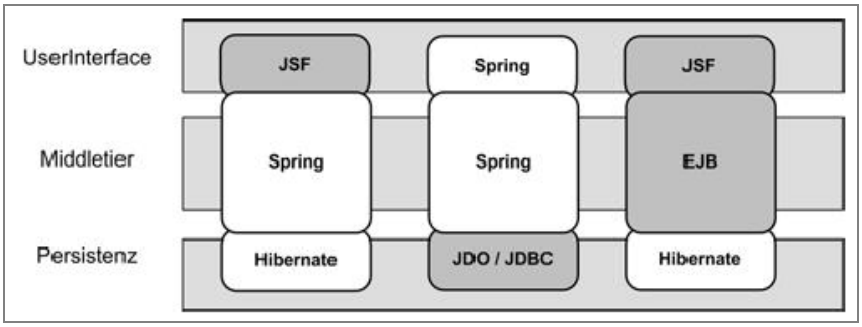


Abbildung 4-2: Kombination Spring und Hibernate – Kombination verschiedener Technologien in Drei-Tier-Architektur, Kombination von Java ServerFaces (JSF), Spring und Hibernate (links), Kombination Spring, Spring und JDBC (mitte) und Kombination von JSF, Enterprise JavaBeans und Hibernate (rechts) [36]

Andererseits ist in Abbildung 4-2 ersichtlich, dass *Spring* auch im Bereich der View (hier UserInterface) integriert werden kann. Es ist aber ebenso möglich, *Spring* mit der

Präsentationstechnologie *JavaServer Faces* (*JSF*) zu kombinieren. Durch diese Entkopplung kann der Entwickler auch in diesem Bereich die Technologie weitgehend frei bestimmen und ist nicht an *Spring* als View-Technologie gebunden.

Auch im Bereich Authentifizierung und Autorisierung bringt *Spring* ein eigenes entwickeltes Framework namens *Spring Security* mit. *Spring MVC* stellt also einen ähnlichen Mechanismus wie *Seam* für die Internationalisierung der Applikation bereit. Die einzelnen Sprachinformationen werden in *Properties-Dateien* ausgelagert und können zur Laufzeit sprachspezifisch angezeigt werden. [39]

Fazit

Beide Frameworks bieten zwar für alle Kriterien gute Implementierungsansätze, jedoch werden gerade unerfahrene Entwickler mit der Masse an Konfigurationsmöglichkeiten unter *Spring MVC* „erschlagen“. Die Konfigurationen während des Programmierprozesses halten sich unter *Seam* durch den mitgelieferten Generator *seam-gen* in Grenzen. Abschließend für die Untersuchung beider Frameworks ist zu erwähnen, dass sich *Seam* speziell in den Kriterien (siehe Tabelle 4-1) Einarbeitungsaufwand und Konfigurationsaufwand klar durchgesetzt hat. Aus diesen Gründen wird es als Web-Framework für die autosomale Datenbank eingesetzt.

	JBoss Seam	Spring MVC
Installation	+	o
Einarbeitungsaufwand	+	-
Dokumentation	++	++
Konfigurationsaufwand	+	-
Hibernate	++	++
Navigation	++	++
Internationalisierung	++	++
Authentifizierung& Autorisierung	++	++
View-Technolgie	++	++

Tabelle 4-1: Bewertung der Frameworks – Kriterien (links), Frameworks (oben), Kriterium erfüllt: ++ sehr gut, + gut, o durchschnittlich, - schlecht, -- sehr schlecht

4.2 JBoss Seam

„Die Java Platform, Enterprise Edition (*Java-EE*) ist in der aktuellen Version 5 ein modernes Komponenten-Framework, das eine ganze Reihe verschiedener Spezifikationen zu einem Ganzen bündelt.“ [33, S.9]
Seam vereint eine Vielzahl dieser Spezifikationen und erleichtert deren Zusammenspiel in einer Anwendung. In diesem Kapitel werden einige Spezifikationen sowie Erleichterungen im Umgang mit ihnen, die mit *Seam* im Zusammenhang stehen, näher erläutert. Diese Zusammenhänge und Spezifikationen finden sich in anwendungsspezifischen Szenarien des Prototyps wieder.

4.2.1 Java ServerFaces Framework

Für die Ausgabe von dynamischen *HTML*-Seiten einer Webanwendung verwendete man im Java Umfeld die rudimentäre Technologie *Java ServerPages* (*JSP*). [22] Diese Technologie ermöglicht die Einbettung von Java-Code und somit dynamischen Inhalt in *HTML*-Seiten. Bei nicht diszipliniertem Vorgehen bei der Erstellung von Programmcodes kann es passieren, dass *HTML*- und Java-Code vermischt werden. Dies führt schnell zu einer unübersichtlichen Programmstruktur, die die Entwicklung deutlich erschwert. Um die Entwicklung von Webanwendungen dennoch weiter zu vereinfachen, wurde im März 2004 *JavaServer Faces* (*JSF*) ins Leben gerufen. [40] *JSF* wird als eine Spezifikation der *Java Enterprise Edition* (*Java EE*) bezeichnet, mit welcher sich Benutzeroberflächen speziell für Webanwendungen erstellen lassen. *JSF* setzt dabei auf die *Servlet*- und *JSP*-Technologie. [22]

Der Programmierer entwickelt dabei die Ausgabeseiten nicht wie gewohnt in *HTML*-Code, sondern programmiert auf einer höheren Abstraktionsebene. Dabei werden bereits fertige Komponenten, welche den *HTML*-Code generieren, in eine Seite eingebunden. Ähnlich wie bei der klassischen Anwendungsentwicklung kann sich der Programmierer auf die eigentliche Programmlogik konzentrieren und muss sich nicht mit grafischen Problemen beschäftigen. Es besteht die Möglichkeit auch eigene Komponenten zu entwickeln. Diese können dann in verschiedensten Webanwendungen Verwendung finden. *JSF* benutzt das Modell-View-Controller-Entwurfsmuster. Somit

werden Daten, Präsentation und Programmsteuerung voneinander getrennt (siehe Kapitel 3.1).

Zusätzlich kann *JSF* durch verschiedene *Komponentenbibliothek*en, wie z.B. MyFaces von Apache, *RichFaces* von JBoss oder IceFaces von Icesoft erweitert werden. Im nächsten Absatz werden allgemeine Informationen zur im Prototyp einsatzfindenden *Komponentenbibliothek RichFaces* näher erläutert.

4.2.2 JBoss RichFaces

Mit JBoss *RichFaces* (ab jetzt *RichFaces*) wird eine *Komponentenbibliothek* für *JavaServer Faces* (*JSF*) bezeichnet. Mit ihr lassen sich *Rich Internet Applications* (*RIAs*) erstellen. Die *Komponentenbibliothek* besteht aus zwei Teilen, der Ajax4JSF-Bibliothek, mit der sich bestehende *JSF*-Komponenten um *AJAX*-Funktionalität (siehe Kapitel 3.5) erweitern lassen, sowie dem *RichFaces*-Teil, welcher vorgefertigte Komponenten mit *AJAX*-Unterstützung bereitstellt. Abbildung 4-3 zeigt eine *RichFaces*-Komponente zum Einstellen des Datums. [41]

Abbildung 4-3: RichFaces Kalender-Komponente – Eine Komponente in RichFaces zum Einstellen des Datums

Um die Funktionalität des Frameworks im vollen Umfang nutzen zu können müssen wie in Abbildung 4-4 ersichtlich, die *RichFaces Taglibrarys* rich in Zeile sechs sowie a4j in Zeile sieben, in eine bestehende *JSF*-Seite implementiert werden. Nach dem Einbinden der *Tag Libraries* sowie dem Import der notwendigen *JAR-Dateien* kann die Funktionalität von *RichFaces* genutzt werden.

```
1. <ui:composition xmlns="http://www.w3.org/1999/xhtml"
2.                 xmlns:s="http://jboss.com/products/seam/taglib"
3.                 xmlns:ui="http://java.sun.com/jsf/facelets"
4.                 xmlns:f="http://java.sun.com/jsf/core"
5.                 xmlns:h="http://java.sun.com/jsf/html"
6.                 xmlns:rich="http://richfaces.org/rich"
7.                 xmlns:a4j="http://richfaces.org/a4j">
8.                 template="layout/template_new.xhtml"
9.                  ...
10. </ui:composition>
```

Abbildung 4-4: RichFaces Tag Libraries – demonstriert das Einbinden der Tag Libraries rich und a4j

Durch die Komplexität von *RichFaces*, sowie der aktiven Community, welche viele Tutorials sowie eine ausführliche Dokumentation im Netz bereitstellt, wurde es als *Komponentenbibliothek* für die Präsentations-Technologie *JavaServer Faces* ausgewählt. [42] [43]

4.2.3 Enterprise JavaBeans und Java ServerFaces unter Seam

Das Zusammenspiel einer *Enterprise JavaBeans*-Komponente (*EJB*-Komponente, siehe Kapitel 3.4) und einer *Java ServerFaces*-Seite (*JSF*-Seite) unter *Seam* wird zum Verständnis im Folgenden anhand eines fiktiven Beispiels erläutert.

Zu Beginn des Programmierprozesses beschäftigt sich der Entwickler in der Regel mit den *Entity Beans* (siehe Kapitel 3.4). Diese repräsentieren die Geschäftsobjekte der Anwendung. Abbildung 4-5 zeigt eine *Entity Bean*. Sie besitzt zwei Klassen-Attribute und die dazugehörigen *Getter*- und *Setter-Methoden*.

```
1. @Entity
2. @Name ("person")
3. public class PersonEntity implements Serializable {
4.
5.    private int id;
6.    private String name;
7.
8.    @Id
9.    @GeneratedValue
10.   public int getId() {
11.         return id;
12.   }
13.   public void setId(int id) {
14.         this.id = id;
15.   }
16.
17.     public String getName() {
18.         return name;
19.   }
20.
```

```
21.    public void setName(String name) {
22.          this.name = name;
23.    }
24. }
```

Abbildung 4-5: Entity Bean – demonstriert den Aufbau einer Entity Bean mit zwei Attributen und den dazugehörigen Getter- und Setter-Methoden

Die *Annotation* @Entity in Zeile eins kennzeichnet die Klasse als *Entity Bean*. In Zeile zwei wird der Klasse durch die *Annotation* @Name der Name person zugeordnet. Unter diesem Stringnamen wird die *Entity Bean* unter *Seam* registriert. In Abbildung 4-6 ist eine *JSF*-Seite zu sehen, welche die obige Bean als Backend des Eingabetextfeldes des Formulars verwendet. Die Notation #{person.name} aus Zeile drei bezieht sich auf das name-Attribut der *Seam*-Komponente person, die eine Instanz der *Entity Bean* PersonEntity ist (siehe Abbildung 4-5). Mit der #{…}-Notation werden Java-Objekte unter *Seam* referenziert. Sie ist ein Ausdruck der *JSF Expression Language* (*EL*), die in *Seam* sehr oft Verwendung findet. [44]

```
1. <h:form>
2.    Bitte geben Sie Ihren Namen ein:<br/>
3.    <h:inputText value="#{person.name}" size="15"/><br/>
4.    <h:commandButton value="Senden" action="#{manager.doSomething}"
5.     type="submit"/>
6. </h:form>
```

Abbildung 4-6: Formular JSF-Seite - demonstriert die Verwendung einer Entity Bean als Backend-Property für die <inputText>-Komponente

Wenn ein Anwender nach der Eingabe seines Namens nun den Button Senden aus Zeile vier anklickt, um das Formular abzusenden, erstellt *Seam* die verwaltete person-Komponente mit den Eingabedaten. Dann ruft es die doSomething-Methode auf. Diese könnte z.B. das person-Objekt in einer Datenbank abspeichern.

Aus dem Beispiel geht hervor, dass mit Hilfe der *Annotations* die *EJB*-Komponenten direkt in der Präsentationsschicht verwendet werden können und durch *Seam* verwaltet werden. Verglichen mit Anwendungen, die mit anderen Web-Frameworks entwickelt wurden, sind *Seam*-Anwendungen konzeptionell einfach und erfordern für dieselbe Funktionalität erheblich weniger Code. [33]

5 Entwurf der autosomalen Datenbank

In diesem Kapitel erfolgt eine Beschreibung zum Entwurf der Webanwendung. Zuerst werden funktionale und nicht funktionale Anforderungen, welche an die Anwendung gestellt werden, definiert, gefolgt von der Softwarestruktur sowie einer *Use Case*-Analyse zu den Hauptfunktionen der autosomalen Datenbank. Abschließend wird das Layout und das Klassenmodell anhand von zwei Beispielen erläutert. Da das Web-Framework *Seam* zum Einsatz kommt welches durch das *Hibernate*-Framework die *Objekt-Relational Mapping*-Funktionalität (*ORM*-Funktionalität) unterstützt, kann in diesem Kapitel auf den Entwurf des Datenbankmodells verzichtet werden. *Objekt-Relational Mapping* ist eine Technik in der Softwareentwicklung, mit der die Klassenobjekte automatisch in eine relationale Datenbank umgewandelt („gemappt“) werden. Der Webanwendung erscheint die Datenbank dann als objektorientierte Datenbank, was die Entwicklung erheblich vereinfacht.

5.1 Anforderungen an die Umsetzung

Da im Kapitel 2.2 bestehende Systeme beleuchtet wurden ist es nun notwendig, die konkreten Anforderungen für die Realisierung festzulegen. Die folgenden Anforderungen wurden auf Basis der aus Kapitel 2.3 gewonnenen Erkenntnisse, sowie aus dem Funktionsspezifikationskatalog der Qualitype AG abgeleitet. [16]

5.1.1 Funktionale Anforderungen

Für eine konkrete Realisierung ist es notwendig, dass die Applikation die folgenden funktionalen Anforderungen erfüllt, um der Aufgabenstellung im vollen Umfang gerecht zu werden und sich somit breitflächig etablieren kann.

Ein **Anzeigeservice** soll die Daten in geeigneter Form übersichtlich über die View präsentieren. Texte, Tabellen und andere Daten wie z.B. Bilder sollen strukturiert zur Übersicht gebracht werden. Durch den Einsatz von auf- und zuklappbaren *Panels* kann der Benutzer, für ihn nicht relevante Informationen ein- oder ausblenden. Die große Anzahl von Daten soll übersichtlich mit Hilfe von Tabellen zur Anzeige gebracht

werden. Um die Übersichtlichkeit zu gewährleisten, sollen diese Tabellen mit *Datascrollern* ausgestattet sein und so dem Benutzer nur eine gewisse Anzahl von Ergebnissen liefern. Mit Hilfe der *Datascroller* kann sich der Benutzer die Ergebnisse Seite für Seite anzeigen lassen, ohne dabei die Seite auf der er sich gerade befindet zu verlassen.

Ebenso soll ein **Suchservice** implementiert werden, welcher in der Lage ist, Parameter vom Benutzer entgegen zu nehmen und diese an die Datenbank weiterzuleiten. Bei erfolgreicher Suche sollen die Daten aus der Datenbank aufbereitet und strukturiert durch den Anzeigeservice zur Anzeige gebracht werden. Es soll auch möglich sein, Suchparameter zu kombinieren (siehe Tabelle 5-1), um so den Wertebereich gefundener Resultate einzugrenzen. Der Suchservice unterstützt den Benutzer dahingehend, dass dieser schnell und ohne Umwege an die gewünschten Daten kommt. Stößt der Suchservice auf mehrere Ergebnisse, werden diese in einer Tabelle ausgegeben und können daraufhin über Hyperlinks angesteuert werden. Bei nur einem Suchtreffer soll das System den Nutzer automatisch auf die Seite mit den gefunden Informationen navigieren. Die nachfolgende Tabelle 5-1 zeigt mögliche Kombinationen von Suchparameter und die eintreffenden Möglichkeiten (Navigation zur jeweiligen Webseite) bei erfolgreicher Suche auf.

Parameter 1	Boolescher Operator	Parameter 2	Navigation
Marker	UND	-	Markerseite
Marker	UND	Chromosom	Markerseite
Marker	UND	Allel	Markerseite
Marker	ODER	Allel	Markerseite
Marker	ODER	Population	Marker-/Populationsseite
Marker	UND	Population	Markerseite
Population	UND	-	Populationsseite

Tabelle 5-1: Verknüpfung Suchparameter - mögliche Kombination von Suchparametern und die einzutretenden Ergebnisse (Navigation zur jeweiligen Webseite)

Um die Webanwendung mit nützlichen Funktionalitäten auszustatten, bietet es sich an, einen **Berechnungsservice** zu implementieren. Dieser bietet Berechnungen an, um eine Aussage über die Qualität der Daten zu treffen. Die einzelnen Werte (Parameter) umfassen dabei:

- *Homozygotie* (*h*)
- *Heterozygotie* (*HET*)
- Power of Exclusion (PE)
- *Power of Discrimination* (*PD*)
- *Paternity Index* (PI)
- *Mean Exclusion Chance (MEC)*
- Polimorphic Information Content (PIC).

Diese Berechnungen sollen dynamisch für jede angezeigte *Population* erstellt werden und durch den Anzeigeservice mit Hilfe einer Tabelle visualisiert werden. Für die Berechnungen sollte nicht der primitive Datentyp *Double* verwendet werden, da dieser Rechnungsungenauigkeiten aufweist. Es ist ratsam den Datentyp *BigDecimal* zu verwenden, da dieser diese Schwäche nicht aufweist. [45]

Ein **Importservice** erlaubt es allen Wissenschaftlern des forensischen Umfeldes Daten über intelligente Eingabemasken in die Datenbank einzupflegen. Diese Eingabemasken sollen die Daten schon im Vorfeld auf Gültigkeit überprüfen. Somit soll die Datenbank frei von Redundanz („Datenmüll“) gehalten werden. Der Übermittlungsservice soll es erlauben mehrere Wege zu unterstützen, wie Daten in die Datenbank kommen sollen. Alternativ zu den Eingabemasken soll ebenfalls ein *CSV-Import* angeboten werden, um dem Benutzer umständliche Eingaben über die Website zu ersparen. Nachdem die Datenübermittlung erfolgreich abgeschlossen ist, soll ein Verantwortlicher der autosomalen Datenbank, über einen Benachrichtigungsdienst via E-Mail, darüber informiert werden. Nach der erfolgreichen Kontrolle von dem Verantwortlichen der Datenbank können die übermittelten Daten freigeschaltet werden.

Da eine Datenbank vom Zugriff der Nutzer lebt, soll es auch eine Möglichkeit geben Datensätze herunterladen zu können. Für diesen Zweck soll ein **Exportservice** die funktionale Anforderungsliste abrunden. Er ermöglicht es, Daten aus der Datenbank in verschiedenen Formaten herunterzuladen und diese für die spätere Verwendung, z.B. als Input für andere Softwareprodukte zur Verfügung zu stellen. Als Standardformat für den Prototyp soll hier das *xls-Format* von Microsoft Excel zum Einsatz kommen, da das Produkt von Microsoft Office deutschlandweit (siehe Abbildung 5-1) die größte

Verbreitung aufweist. Später soll auch ein Export für das Dateiformat des Produktes *Calc* von Open Office implementiert werden, da es die zweitstärkste Verbreitung in der Statistik aufweist und zudem noch ein Open Source Produkt ist. Ebenso sollen noch die Exportierungsformate PDF und *XML* angeboten werden, da sie mit den nötigen Plugins direkt im Web-Browser angezeigt werden können und somit nicht zwangsläufig ein eigenes Softwareprodukt für die Anzeige voraussetzen.

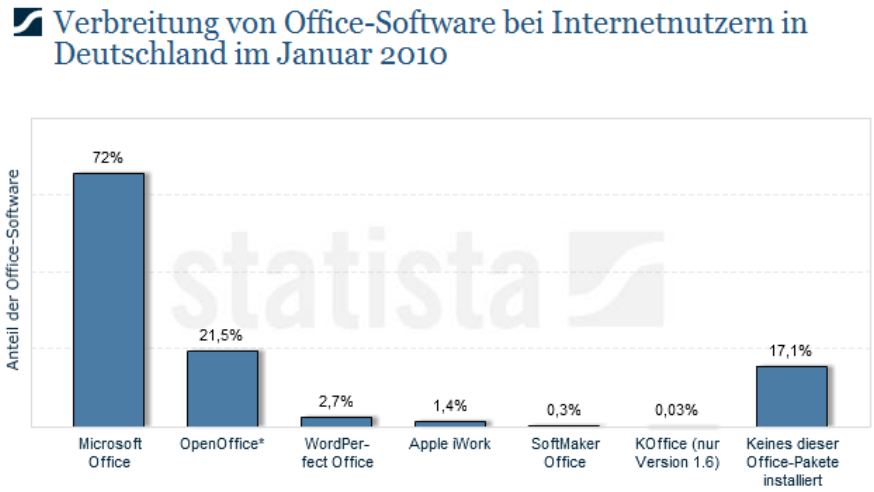


Abbildung 5-1: Verbreitung Office-Software – zeigt die prozentuale Verbreitung von Office-Software in Deutschland [46]

All diese Komponenten sollen sich in der geplanten Architektur wieder finden, um ein serviceorientiertes System zu bieten und beliebig erweitert werden kann. Diese Services stellen die fundamentale Grundlage für das System dar.

5.1.2 Nicht funktionale Anforderungen

Unabhängig von den funktionellen Anforderungen des Systems, sollen die nicht funktionellen Anforderungen das System zusätzlich in Qualität und Leistung erweitern und unterstützen. Die nicht funktionalen Anforderungen gelten dabei als sehr wichtige Festlegung, da mit ihnen die Richtlinien bei der Erstellung der Software definiert werden. Im Folgenden wird auf fünf spezielle Anforderungen eingegangen.

Die **Performance** soll mit ausgereiften und aktuellen Technologien das Maximum aus der Anwendung herausholen. Ziel der Performancesteigerung ist es, schnell und leistungsstark auf Anfragen vom Benutzer reagieren zu können. Der eingesetzte

Applikationsserver soll mit optimaler Konfiguration eingerichtet werden und dies so gewährleisten.

Portabilität oder auch Plattformunabhängigkeit soll durch die eingesetzte Technologie Java gewährleistet werden. Da Java mit der eigenen Umgebung (*Java Runtime Environment*) auf allen gängigen Systemen arbeiten kann, bietet es sich hier gut für die Umsetzung an. Da die autosomale Datenbank in erster Linie über eine Website erreichbar sein soll, wird das World Wide Web (WWW) als Übertragungsmedium benutzt. Dies bietet sich an, da das einheitliche *Hyper Text Transfer Protokoll* (*HTTP*) auf allen Systemen Einsatz findet.

Durch das JBoss *Seam* Security Framework (*Rules*) soll es möglich sein ein hohes Maß an **Sicherheit** zu bieten. In ihm werden komplexe Regeln definiert. Hier lässt sich festlegen wer welche Seite einsehen kann oder auf welche *UserInterface-Komponente* (UI-Komponente) oder welche *JavaBean*-Methode zugegriffen werden darf. Wie alles andere in *Seam* sind auch die *Seam* Sicherheitsregeln zustandsbehaftet. Das bedeutet, dass das Ergebnis einer Regel vom jeweiligen Anwendungskontext abhängt. Damit ist es möglich, bestimmten Anwendern den Zugriff auf bestimmte Anwendungsfunktionen nur dann zu gestatten, wenn bestimmte Laufzeitbedingungen erfüllt sind. [31]

Die Website soll prinzipiell **mehrsprachig** in Englisch und Deutsch angeboten werden. Im Ausblick auf später soll eine Spracherweiterung auf Spanisch und Französisch ohne großen Aufwand möglich sein. Dies soll es ermöglichen ein breites Spektrum an Wissenschaftlern auf der ganzen Welt anzusprechen.

Um die Anzahl an Benutzern der Webanwendung zu maximieren, sollen durch Strukturierung, Übersichtlichkeit und moderne View-Technologien der Webanwendung ein hohes Maß an **Benutzerfreundlichkeit** geboten werden. Die Anwender sollen schnell einen Überblick über alle Funktionalitäten erlangen. Dies soll durch ein *DropDown-Menü*, welches auf jeder Seite egal wo man sich in der Webanwendung befindet, im Kopfbereich sichtbar sein. Eine Hilfefunktion soll zum schnellen Verständnis der Anwendung beitragen.

5.2 Softwarearchitektur

Nachdem alle Anforderungen an das System definiert sind, ist der nächste Schritt das konkrete Design der Softwarearchitektur festzulegen. Da die Architektur aus mehreren verteilten Komponenten besteht, wird zuerst ein Überblick über das gesamte System gegeben. Später wird im Kapitel 6.1.3 näher auf das Datenbankmanagementsystem *PostgreSQL* eingegangen.

Überblick über die gesamte Architektur
Die Architektur der autosomalen Datenbank soll in eine *Drei-Tier-Architektur* eingebettet werden (siehe Abbildung 5-2). [47]

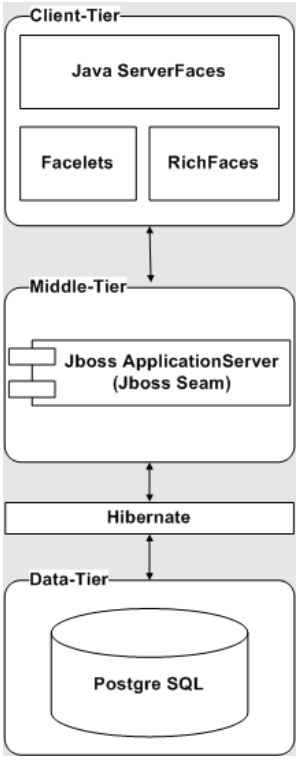


Abbildung 5-2: Architektur autosomalen Datenbank - Architektur der autosomalen Datenbank in Drei-Tier-Architektur, Java ServerFaces, Facelets, RichFaces Technologien Client-Tier, JBoss Application Server und Seam Middle-Tier und PostgreSQL als Datenbankmanagementprogramm in Data-Tier, Hibernate als Schnittstelle zwichen Middle-Tier und Data-Tier

Die Client-Tier dient zur Präsentation der Daten. Als Technologien in dieser Schicht wurde *Java ServerFaces*, *RichFaces* und *Facelets* ausgewählt (siehe Kapitel 4.21, 4.22 und 6.2.2). Die Visualisierung der Daten erfolgt in dieser Schicht in der Regel über einen Web-Browser. Um die Benutzerfreundlichkeit der Anwendung zu steigern, ist es vorteilhaft wenn der Web-Browser mit der JavaScript- und *AJAX*-Technologie ausgestattet ist. Die Client-Tier steuert das Benutzer-Interface, nimmt Benutzer-Events

entgegen und leitet diese an die Middle-Tier weiter. Diese Schicht wird in der autosomalen Datenbank durch den *JBoss Application Server* repräsentiert. In diesem ist die eigentliche Webanwendung, die in unserem Fall durch das Web-Framework *Seam* umhüllt wird, enthalten. *Seam* beinhaltet die gesamte Geschäftslogik und stellt der darüberliegenden Schicht mit Hilfe des *Applikationsserver* diese Dienste zur Verfügung. Die Data-Tier, welches die letzte und unterste Ebene unserer Architektur darstellt, ist für die Persistierung der Daten zuständig. Sie besitzt selbst keine Logik und wird durch die Middle-Tier gesteuert. Für die autosomale Datenbank wird hier das Datenbanksystem *PostgreSQL* (siehe Kapitel 6.1.3) eingesetzt. Als Schnittstelle zwischen Middle- und Data-Tier kommt *Hibernate* zum Einsatz. Es unterstützt den Entwickler, um die Geschäftsobjekte in einer relationalen Datenbank abzubilden.
Durch den Einsatz der *Drei-Tier-Architektur* wird das System komplett entkoppelt. Somit ist es z.B. möglich das Datenbanksystem ohne eine Veränderung der Programmlogik auszutauschen.

5.3 Use Cases

Die verschiedenen Anwendungsfälle der Software wurden anhand der UML-Standardisierung entworfen. [48] In Abbildung 5-3 ist ein *Use Case*-Szenario dargestellt, welches die Hauptfunktionen der autosomalen Datenbank zeigt. Weitere *Use Cases* befinden sich im Anhang Anlagen Teil 2.

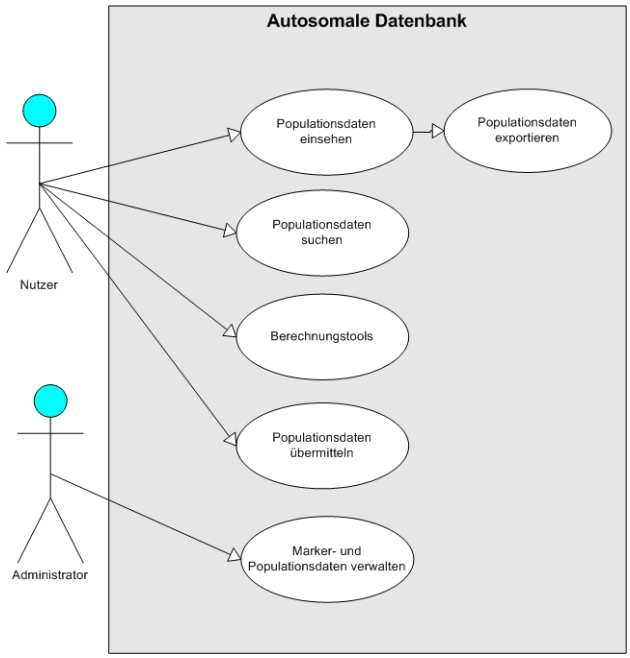


Abbildung 5-3: Use Cases autosomale Datenbank - Allgemeine Hauptfunktionen der Autosomalen Datenbank

Wie in Abbildung 5-3 ersichtlich ist, gibt es zwei Möglichkeiten mit dem System zu interagieren. In der Rolle des Nutzers bietet sich dem Anwender ein serviceorientiertes System. Er hat die Möglichkeit Populationsdaten einzusehen und anschließend zu exportieren, nach ihnen zu suchen, diverse Berechnungswerkzeuge zu nutzen oder selbst Populationsdaten an das System zu übermitteln. Auf der anderen Seite steht die Rolle des Administrators. Er kann vorhandene Populations- oder Markerdaten verwalten und Änderungen am System bestätigen.

5.4 Layout

Da die autosomale Datenbank als Webanwendung entwickelt wird, soll das Design wie bei einer gängigen Website gestaltet werden. Das Layout wird in drei Teilbereiche unterteilt (siehe Abbildung 5-4). Der Kopfbereich, in ihm sollen sich ein Navigationsmenü sowie ein Logo wiederfinden. Weitere Funktionalitäten wie eine Sprachauswahl und eine Schnellsuche sollen ebenfalls im Kopfbereich Platz finden. Im Zentrum der Seite befindet sich der Contentbereich. Je nach Informationsseite sollen die Daten dort in einem Haupt-*Panel* angezeigt werden. Im rechten Bereich wird ein Informationsbereich positioniert. Neuigkeiten und Statistiken zur Webanwendung sollen hier in verschiedenen *Panels* zur Anzeige gebracht werden (siehe Abbildung 5-4).

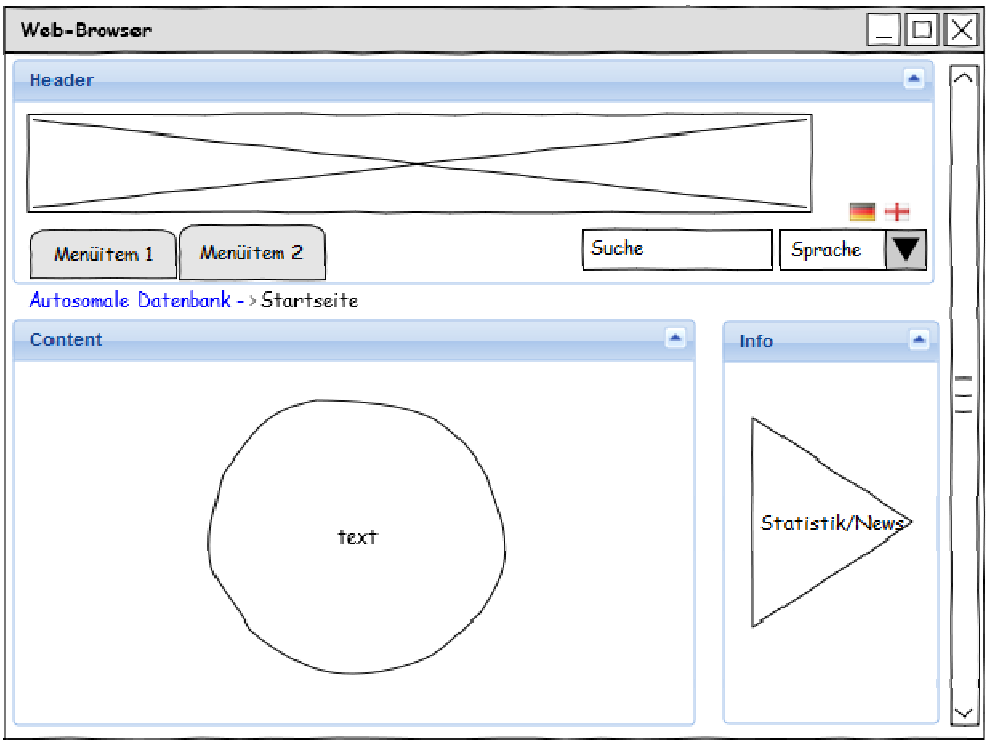


Abbildung 5-4: Layoutentwurf autosomale Datenbank - erstellt mit Evolus Pencil 1.2

5.5 Klassenmodell

Der Klassenentwurf der Anwendung resultiert aus den spezifischen Anforderungen von denen im Folgenden zwei Beispiele näher erläutert werden. Die verwendeten Abbildungen sind Ausschnitte des in den Anlagen Teil 3 befindlichen Klassenmodells.

Assoziation „Marker“, „Synonym“ und „MarkerReference“

Weil die Bezeichnung der einzelnen Markertypen weltweit stark variiert, sollten einem *Marker* mehrere Synonyme zugeordnet werden können, um so zur besseren Widerauffindbarkeit beitragen. In Abbildung 5-5 sieht man diese Relation in der UML-Standardisierung. Ein beliebiger *Marker* besitzt mehrere Synonyme. Hierbei handelt es sich um eine „1 zu n“ Beziehung. Des Weiteren können einem *Marker* mehrere Referenzen zugeordnet werden. Auch diese Relation wird mit einer „1 zu n“ Beziehung realisiert. Die Beziehungen sind bidirektional, dass heißt jedes Objekt „Synonym“ kennt das dazugehörige Objekt „Marker“. Im Umkehrschluss bedeutet dies, dass jedes Objekt „Marker“ die Existenz seiner dazugehörigen Objekte „Synonym“ kennt.

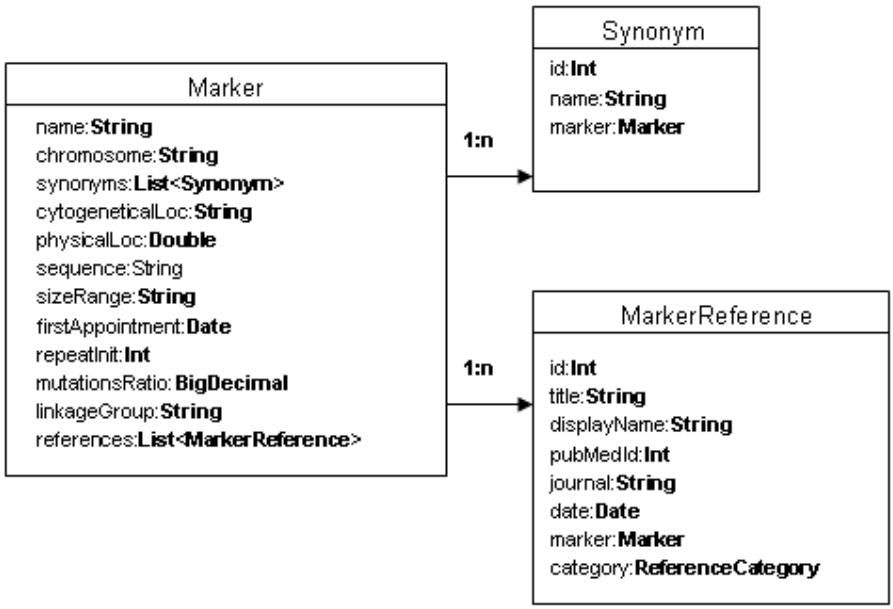


Abbildung 5-5: Ausschnitt Klassenmodell 1 - Assziation „Marker“, „Synonym“ und „MarkerReference“

Assoziation „Population“, „MarkerFrequency” und “Frequency”

Ein Populationsdatensatz besteht aus einer Grundmenge von *Allelfrequenzen* und einer dazugehörigen *Population* für die diese Daten vorliegen. Wie in Abbildung 5-6 ersichtlich werden die einzelnen *Allelfrequenzen* durch die Klasse „Frequency“ repräsentiert. Ein Populationsdatensatz, hier „MarkerFrequency“, ist mit mehreren Klassen-Objekten „Frequency“ referenziert. Die Beziehung der beiden Klassen wird durch eine „1 zu n“ Beziehung realisiert. Der Datensatz wird letztendlich noch durch eine „n zu 1“ Beziehung mit der Klasse „Population“ verknüpft. Daraus ergibt sich, dass eine *Population* mehrere Populationsdatensätze besitzen kann.

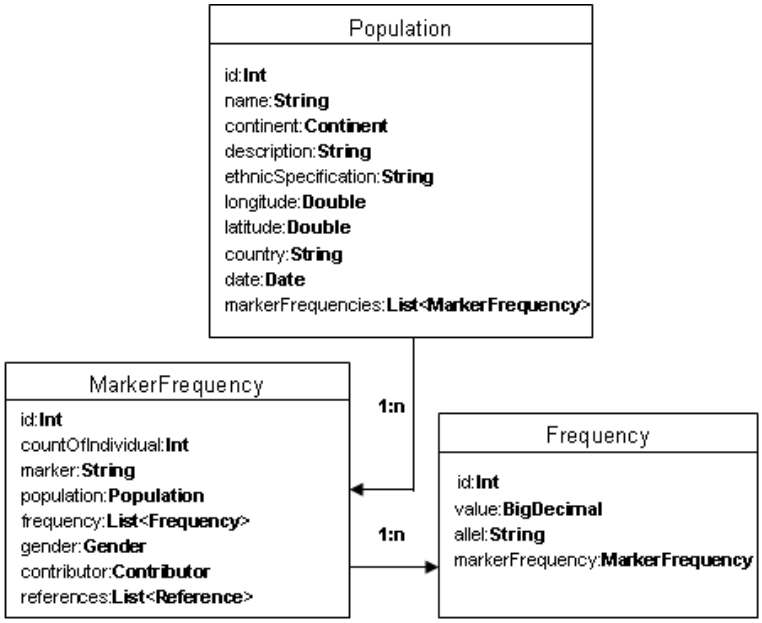


Abbildung 5-6: Ausschnitt Klassenmodell 2 - Assoziation „Population“, „MarkerFrequency“ und „Frequency“

6 Implementierung des Prototyps

6.1 Entwicklungswerkzeuge und Technologien

Für die Entwicklung komplexer verteilter Webanwendungen steht dem Programmierer eine Vielzahl an Werkzeugen zur Verfügung. Die meiste Bedeutung kommt hier der *Integrated Development Environment* (*IDE*) zu. Sie nimmt dem Entwickler häufig wiederkehrende Aufgaben ab und sorgt für einen schnellen Zugriff auf wichtige Funktionen. Eine leistungsstarke *IDE* unterstützt den Entwickler außerdem im Bereich Versionsverwaltung, *Syntax Highlithing*, *Debugging* und vielen mehr. [49]

6.1.1 Eclipse IDE

Eine *IDE*, welche die oben genannten Kriterien im vollen Umfang erfüllt, ist die Entwicklungsumgebung *Eclipse* Galileo 3.5 von der *Eclipse* Foundation. [50] Zum einen ist sie kostenlos verfügbar und darüber hinaus bietet sie die Möglichkeit durch den Einsatz von Plugins den Funktionsumfang zu erweitern. Explizit für die Entwicklung der autosomalen Datenbank wurde das Plugin *Subclipse* nach [51] in die Entwicklungsumgebung integriert. [52] Es verbindet Eclipse mit der Versionsverwaltung *Subversion*. [53] Bei *Subversion* handelt es sich um ein System zur Versionierung und zur Kontrolle auf den gemeinsamen Zugriff von Dokumenten und Dateien. Der Programmcode wird dabei auf einem zentralen Server gelagert. Damit ist es jederzeit möglich eine bestimmte Version des Projektes wiederherzustellen. Der Einsatz eines solchen Werkzeuges bringt viele Vorteile mit sich, vom Einarbeitungsaufwand abgesehen.

Durch das *JBoss Tools* Plugin wurde der Funktionsumfang zusätzlich erweitert. [54] Es ist unter anderen mit einem *VisualEditor* für *Java ServerFaces* (*JSF*) ausgestattet. [42] Mit diesem kann man eine *JSF*-Seite in der *IDE* begutachten ohne das Projekt bauen (deployen) zu müssen. Außerdem werden bei den *JBoss Tools* Werkzeuge für den *JBoss Application Server* mitgeliefert. Diese bringen einen Servernavigator zum Steuern und Überwachen des *Applikationservers*, sowie Hilfestellungen beim *Packaging* und *Deployment* mit. Auf den *JBoss Application Server* wird im nachfolgenden Absatz noch einmal näher eingegangen.

6.1.2 JBoss Application Server

Um die Vorteile von JBoss *Seam* und hier speziell die Kombination zwischen *Java ServerFaces* (*JSF*) und *Enterprise JavaBeans* (*EJB*, siehe Kapitel 3.4) vorteilhaft nutzen zu können, ist es notwendig einen *Applikationsserver*, in dem die Webanwendung läuft, einzusetzen. Während der Entwicklungsphase des Prototyps wurde mit dem *JBoss Application Server* (*JBoss AS*) 5.1.0 gearbeitet. Der *JBoss AS,* ist ein Softwareprodukt von der Firma Red Hat, ist neben BEA Weblogic und IBM Websphere einer der drei verbreitetsten *Java EE-Applikationsserver*. [55]
Er besitzt einen modularen Aufbau. Dienste und Anwendungen können zur Laufzeit hinzugefügt oder entfernt werden. Hierzu dient ein spezielles *Deployment*-Verzeichnis in der Verzeichnisstruktur des Servers. Bevor die Entwicklung begann, wurde der *Applikationsserver* nach [56] in die Entwicklungsumgebung integriert. Bis auf die Konfiguration einiger Einstellungen konnte der Server direkt nach der Installation verwendet werden. Während der Entwicklung wirkte es sich negativ aus, dass der Server, in Abhängigkeit von der Rechnerleistung, zwischen 15 Sekunden und über einer Minute zum Starten benötigte. Besonders bei der Entwicklung war dies als sehr nachteilig zu bewerten.

6.1.3 PostgreSQL

Wie in fast jeder Webanwendung benötigt auch die autosomale Datenbank eine Möglichkeit um die Vielzahl an Daten zu speichern. Wie aus Kapitel 5.2 hervorgeht wird für die Persistierung der Daten das Datenbanksystem *PostgreSQL* verwendet. Es ist ein objektrelationales Datenbankmanagementsystem, dass als Open-Source-Programm unter [57] frei verfügbar ist und ohne Lizensierung heruntergeladen und benutzt werden darf. Es wird als das fortschrittlichste Open-Source-Datenbanksystem bezeichnet. *PostgreSQL* unterstützt die SQL92 und SQL99 Standards und bietet zusätzlich eigene Erweiterungen an. [58] Als objektrelationales Datenbanksystem implementiert *PostgreSQL* die Speicherung nicht atomarer Daten, Vererbung und Objektidentitäten und erlaubt Benutzern das System um selbstdefinierte Datentypen, Operatoren und Funktionen zu erweitern. Während der Entwicklung wurde das Datenbankmanagementsystem in Kombination mit der grafischen Administrationsoberfläche *pgAdminIII* benutzt. [59] Dieses Tool erleichtert die Administration der Datenbank erheblich.

6.2 Der Prototyp

Dieses Kapitel gibt einen Überblick über die Funktionalitäten des Prototyps. Am Anfang wird die Projektstruktur unter der *Eclipse IDE* erläutert. Danach wird auf das Layout der Webanwendung, sowie Technologien mit denen der Entwickler bei der Implementierung konfrontiert wurde näher eingegangen. Zum Schluss werden einige Funktionalitäten anhand von Quellcode-Beispielen näher erläutert.

6.2.1 Projektstruktur

Wie schon in Kapitel 4.1.2 beschrieben wird das Java Framework *Seam* für die Implementierung des Prototyps verwendet. Dieses gibt eine bestimmte Ordnerstruktur für die Webanwendung vor. Abbildung 6-1 zeigt diese Ordnerstruktur, die durch den von *Seam* mitgelieferten Generator *seam-gen*, für die Eclipse *IDE* generiert wurde. In den nachfolgenden Zeilen werden die wichtigsten Ordner und deren dazugehörige Inhalte stichpunktartig erläutert.

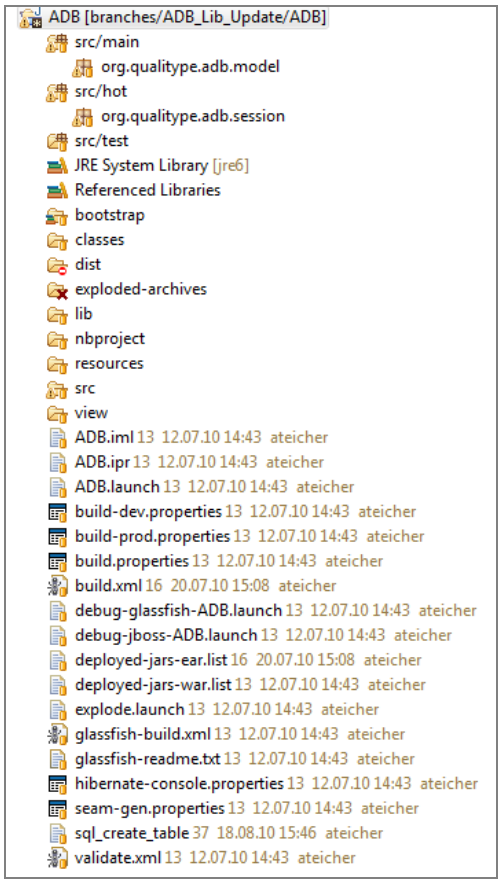


Abbildung 6-1: Projektstruktur autosomale Datenbank – Entwicklungsstruktur der autosomalen Datenbank in der Eclipse-IDE

- **ADB [root]:** beinhaltet alle nachfolgend aufgezählten Ordner, sowie verschiedene *Seam*-Konfigurationsdateien
- **src/main (paket org.qualitype.adb.model):** beinhaltet die *Entity Beans* der Anwendung
- **src/hot (paket org.qualitype.adb.session):** beinhaltet die *Session Beans* der Anwendung
- **src/test:** beinhaltet Klassen für Testfälle
- **JRE System Library:** beinhaltet alle benötigten Referenzdateien für die *Java Runtime Environment* (*JRE*)
- **Referenced Libraries:** beinhaltet Referenzdateien von *Seam*, *Hibernate*, *Drools*, *RichFaces* etc., sowie die qtcommons.jar und qtgenetics.jar von der Qualitype AG
- **exploded-archives:** beinhaltet alle beim Deployment einer *EAR*- oder *WAR-Datei* erstellten Dateien (Konfigurationsdateien, *JSF*-Seiten etc.)
- **lib:** beinhaltet alle *JAR*-Bibliotheksdateien
- **resources:** beinhaltet Hibernate- und Navigationskonfigurationsdateien, sowie *Properties-Dateien* für die Internationalisierung
- **view:** beinhaltet alle *JSF*-Seiten, Bilder sowie *CSS*-Dateien und das Template

6.2.2 Layout und Template

Das Layout der Website wurde gemäß den Vorgaben des Entwurfes aus Kapitel 5.4 übernommen und mit der *Facelets*-Technologie realisiert. *Facelets* ist eine Deklarationssprache zur Beschreibung der Struktur von Seiten unter *Java ServerFaces* (*JSF*). Die Seitenstruktur wird dabei im *XHTML*-Format erstellt und mit *JSF*- und *Facelets*-spezifischen Tags angereichert. *Facelets* bringen dabei einen wirksamen Template-Mechanismus der sehr einfach zu verwenden ist mit. Weiterhin bietet die Deklarationssprache noch weitaus mehr Vorteile, auf die aber hier nicht näher eingegangen werden soll. Dieser Absatz soll nur verdeutlichen, wie mit Hilfe dieser Technologie das Grundtemplate der autosomalen Datenbank erstellt wurde. Alle Unterseiten der Anwendung folgen den Regeln des in Abbildung 6-2 dargestellten Templates.

```
1. <body>
2.     <div id="outer">
3.         <div id="bar" align="center">
4.         </div>
5.         <div id="header">
6.             <ui:include src="menuTop.xhtml" />
7.         </div>
8.         <div id="bodyblock" align="right">
9.             <div id="content" align="center">
10.                <ui:insert name="content"/>
11.            </div>
12.            <div id="info">
13.                <ui:insert name="info"/>
14.            </div>
15.        </div>
16.   </div>
17.</body>
```

Abbildung 6-2: Quellcodeausschnitt aus template.xhtml – demonstriert den Aufbau einer Template-Datei

Es ist ersichtlich, dass Abbildung 6-2 mehrere Div-Bereiche beinhaltet. Das *Div-Tag* ist ein Behälter für weitere *HTML*-Elemente. [60] Drei von ihnen werden zur Laufzeit dynamisch mit Inhalt gefüllt. Im *Div-Tag* in Zeile fünf bis sieben wird die *XHTML*-Seite, welche das Navigationsmenü beinhaltet, eingefügt. Diese Seite kann mit dem *Include-Tag*, aus der *Facelets*-Tag-Reference, eingebunden werden, weil sich der Kopfbereich der Website nie ändert und stets angezeigt wird. In Zeile neun bis elf wird der Content-Bereich eingebunden. Dies erfolgt hier jedoch über das *Insert-Tag*, ebenfalls aus der *Facelets*-Tag-Reference. Dies ist wichtig, weil der Inhalt je nach aufgerufener Webseite dynamisch eingebunden werden soll. Nach demselben Verfahren wird in Zeile zwölf bis vierzehn der Informationsbereich eingefügt. Auch dieser Teil der Anzeige kann je nach angezeigter Informationsseite dynamisch mit Inhalt gefüllt werden. Jede *XHTML*-Seite (*JSF*-Seite) der autosomalen Datenbank verwendet dieses Grundgerüst und muss es im Kopfbereich implementieren. Wie dies realisiert wurde, sieht man in Abbildung 6-3. Alle *JSF*-Seiten der Anwendung sind vom *Composition-Tag* umhüllt. Dieses muss bei der Verwendung von Templates immer eingebunden werden, um den Template-Mechanismus nutzen zu können. Das *Composition-Tag* besitzt den Parameter template. In diesen muss nun die *XHTML*-Datei welche das obige Grundgerüst beinhaltet, implementiert werden. Diese einzelnen Schritte müssen durchgeführt werden, damit die einzelnen *JSF*-Seiten den Regeln des Templates folgen.

```
1.    <!DOCTYPE composition PUBLIC "-//W3C//DTD XHTML 1.0
2.     Transitional//EN"
3.     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4.    <ui:composition xmlns="http://www.w3.org/1999/xhtml"
5.                    xmlns:s="http://jboss.com/products/seam/taglib"
6.                    xmlns:ui="http://java.sun.com/jsf/facelets"
7.                    xmlns:f="http://java.sun.com/jsf/core"
8.                    xmlns:h="http://java.sun.com/jsf/html"
9.                    xmlns:rich="http://richfaces.org/rich"
10.                   xmlns:a4j="http://richfaces.org/a4j"
11.                   xmlns:c="http://java.sun.com/jstl/core"
12.                   template="layout/template.xhtml">
```

Abbildung 6-3: Quellcodeausschnitt aus addPopulation.xhtml – deomostriert die Einbindung des Templates auf einer Webseite

Um nun gezielt die Div-Bereiche mit Inhalten anzureichern, wird das *Define-Tag* verwendet. Es muss den Quellcode, welcher die Anzeigeelemente generiert, die später im Div-Bereich angezeigt werden sollen, umschließen. Diese Vorgehensweise ist in Abbildung 6-4 ersichtlich.

```
1.  <ui:define name="content">
2.     <rich:panel style="width:840px">
3.          <f:facet name="header">
4.                <h:outputText value="Home"/>
5.          </f:facet>
6.          <div style="text-align:justify;">
7.          ...
8.          </div>
9.    </rich:panel>
10. </ui:define>
```

Abbildung 6-4: Quellcodeausschnitt aus home.xhtml – demonstriert die Verwendung des Define-Tags zur Anreicherung von Inhalt einer Webseite

Der Inhalt der home.xhtml-Seite wird dann dynamisch im Content-Bereich des Templates (Abbildung 6-2, Zeile 10) eingebunden.

Das Template definiert also die Richtlinien für das Layout. Die drei Bereiche des Layouts Header, Content und Info finden somit in der Gesamtansicht wie Abbildung 6-4 zeigt ihren Platz. Weitere Browser-Screenshots befinden sich in den Anlagen Teil 4.

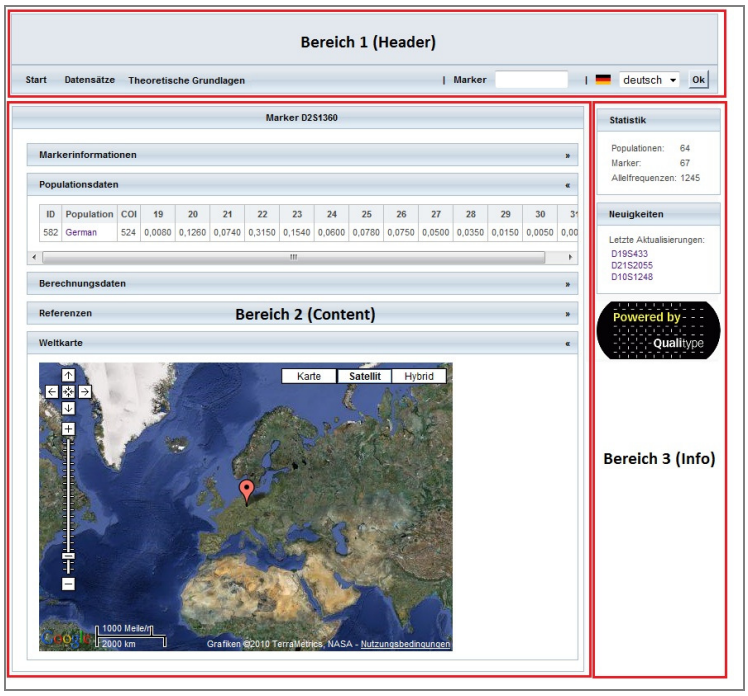


Abbildung 6-5: Layoutansicht - demonstriert die Ansicht der Webseite unter Verwendung eines definierten Templates

6.2.3 Funktionalitäten

In den Folgenden Absätzen werden einige der implementierten Funktionalitäten der Anwendung und speziell deren Umsetzung mit Hilfe von Quelltextausschnitten und zusätzlichen Grafiken (Diagramme, mathematische Formeln und Web-Browser-Ausschnitten) erläutert. Jede Beschreibung einer Funktionalität wird durch ein kurzes Fazit abgerundet.

6.2.3.1 Berechnungsservice

Wie aus dem Kapitel 5.1.1 bekannt ist, soll für die autosomale Datenbank ein Berechnungsservice implementiert werden, welcher die in Kapitel 2.1.2 aufgeführten Berechnungsparameter generiert. Der Service wurde in der Anwendung durch eine einfache Java-Klasse (Abbildung 6-6) realisiert.

```
1. public class Calculation implements Serializable, CalculationImpl
2.  {
3.    private static final long serialVersionUID = 1L;
4.
5.    public BigDecimal getPic( Allele[] alleleFreq ) {...}
6.    public BigDecimal getHet( Allele[] alleleFreq ) {...}
7.    public BigDecimal getMeckrueger( Allele[] alleleFreq ) {...}
8.    public BigDecimal getH( Allele[] alleleFreq ) {{...}
9.    public BigDecimal getPI( Allele[] alleleFreq ) {{...}
10.   public BigDecimal getPE( Allele[] alleleFreq ) {{...}
11.   public BigDecimal getPD( Allele[] alleleFreq) {{...}
12. }
```

Abbildung 6-6: Quellcodeausschnitt aus Calculation.java – demonstriert den Aufbau in Form einer Java-Klasse des implementierten Berechnungsservice

Mit der Notation in Zeile eins wird wie in Java üblich eine Klasse mit dem Namen Calculation erzeugt. Weiterhin ist zu erkennen, dass die Klasse die beiden *Interfaces* Serializable und CalculationImpl implementiert und deren Methoden erbt.

Die Methoden für die biostatistischen Berechnungen (Kapitel 2.1.2) sind in Zeile fünf bis elf zu sehen. Jede der einzelnen Methoden besitzt den Rückgabetyp *BigDecimal*. Ebenso hätte hier auch ein anderer primitiver Datentyp ausgewählt werden können. Aus den funktionellen Anforderungen in Kapitel 5.1.1 geht allerdings hervor, dass z.B. der Datentyp *Double* Rechnungsungenauigkeiten aufweist. Außerdem ist zu erkennen, dass jede Berechnungs-Methode einen Parameter vom Typ Allele[] erwartet. Die Datenstruktur Allele besteht aus dem Namen und dem Frequenzwert eines *Allels*. Jede Methode bekommt also ein Array dieser Datenstruktur übergeben und liefert daraufhin dem Benutzer einen Dezimalwert vom Typ *BigDecimal*. Der Quelltext in Abbildung 6-7 zeigt den Algorithmus zur Berechnung der *Heterozygotie* (*HET*).

```
1. public BigDecimal getHet( Allele[] alleleFreq ) {
2.
3.    BigDecimal sum1 = BigDecimal.ZERO;
4.
5.    for ( int i = 0; i < alleleFreq.length; i++ ) {
6.          sum1 = sum1.add(alleleFreq[i].getAlleleValue().pow(2));
7.        }
8.
9.    return BigDecimal.ONE.subtract(sum1);
10.   }
```

Abbildung 6-7: Quellcodeausschnitt aus Calculation.java 2 – demonstriert den Algorythmus zur Berechnung der Heterozygotie (HET) aus dem Berechnungsservice

In Zeile drei wird eine Variable namens sum1 vom Typ *BigDecimal* instanziert und mit dem Wert null belegt. Danach wird in Zeile fünf bis sieben das Array mit den *Allelen* in einer For-Schleife durchlaufen. Iterativ werden nun, in Abhängigkeit der Länge des

Arrays, die einzelnen Frequenzwerte, der *Allele*, im Quadrat zu der Variable sum1 hinzuaddiert. Zum besseren Verständnis zeigt Formel 6-1 diesen Algorithmus als mathematische Formel.

$$HET = 1 - \sum_{i=1}^{n} p_i^{\,2} = 1 - h$$

Formel 6-1: Berechnung der Heterozygotie - pi=Frequenz des i-ten Allels in der Population, n=Anzahl der untersuchten Personen, h=Homozygotie [14]

Um den Berechnungsservice nun nutzen zu können, muss zuerst ein Objekt der Klasse Calculation, wie Abbildung 6-8 zeigt, instanziert werden.

```
1. Calculation calc = new Calculation();
```

Abbildung 6-8: Quellcodeausschnitt aus MarkerAction.java – demonstriert die Instanzierung eines neuen Objektes der Klasse Calculation (Berechnungsservice)

Anschließend kann nun das zur Demonstration beitragende Objekt populationSet aus Abbildung 6-9, welches einen Datensatz mit den dazugehörigen Klassenattributen (*Population*, Geschlecht, getestete Individuen, etc.) repräsentiert, seine Klassen-Methode setHet aufrufen. Diese *Setter-Methode* setzt einen *Heterozygotie*-Wert für den Datensatz und bedient sich dabei am Berechnungsservice, da es als Parameter das instanzierte Objekt calc, also unseren Berechnungsservice, mit der Methode getHet (Berechnung der *Heterozygotie*) übergibt. Die Methode getHet des instanzierten calc-Objektes bekommt wie oben erwähnt als Parameter den Rückgabetyp (Allele[]) der Methode getAlleles vom Objekt populationSet geliefert.

```
1. populationSet.setHet(calc.getHet(populationSet.getAlleles()));
```

Abbildung 6-9: Quellcodeausschnitt aus MarkerAction.java – demonstriert den Aufruf des Berechnungsservice (getHet) des instanzierten Objektes calc

Der berechnete Wert kann dann ganz einfach auf einer Anzeigeseite ausgegeben werden. In Abbildung 6-10 wird dies mit dem *DataTable-Tag* (Tabelle) aus der *RichFaces-Komponentenbibliothek* bewerkstelligt. Auf die einzelnen Zeilen zur Generierung der Tabelle soll hier nicht eingegangen werden. In Zeile acht bis elf findet die eigentliche Ausgabe des berechneten Wertes statt. In Zeile neun und zehn wird mit dem *ConvertNumber-Tag* der berechnete Wert zur besseren Anzeige formatiert, sodass nur vier Dezimalstellen angezeigt werden.

```
1.  <rich:dataTable var="calc" value="#{populationTablePool}"
2.  id="calculationtable">
3.    ...
4.    <rich:column>
5.     <f:facet name="header">
6.       <h:outputText value="HET" />
7.     </f:facet>
8.     <h:outputText value="#{calc.het}">
9.     <f:convertNumber type="number" maxFractionDigits="4"
10.    minFractionDigits="4" groupingUsed="false" />
11.    </h:outputText>
12.    </rich:column>
13.    ...
14. </rich:dataTable>
```

Abbildung 6-10: Quellecodeausschnitt aus showMarker.xhtml – demonstriert die Ausgabe des berechneten Wertes für die Heterozygotie in einer JSF-Seite

Der Quelltext aus Abbildung 6-10 führt dann zur in Abbildung 6-11 gezeigten Ausgabe im Web-Browser.

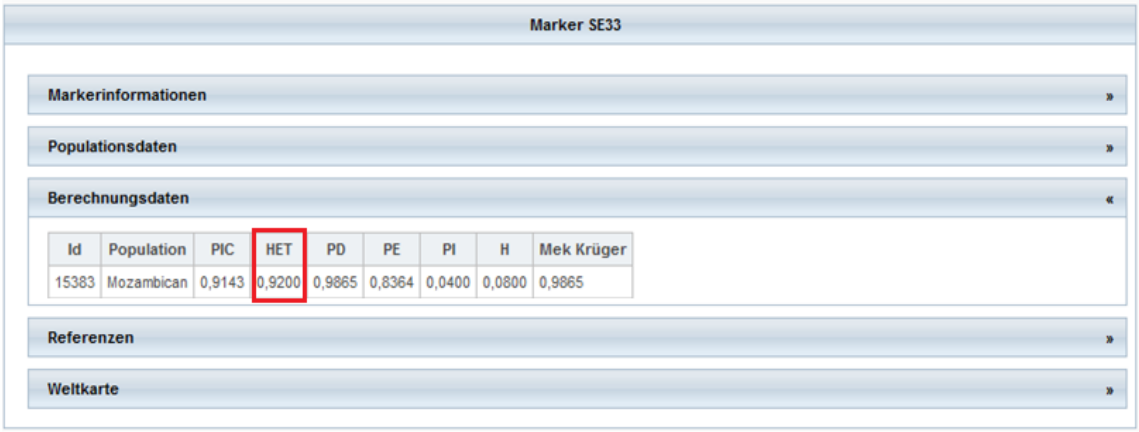


Abbildung 6-11: Berechnungsservice – Ausgabe der Heterozygotie (HET)

Fazit

Ziel der Implementierung von biostatistischen Berechnungen war es, den Benutzern auch Informationen über die Qualität der Datensätze zu liefern. Keine der untersuchten Webanwendungen aus Kapitel 2.2.2 haben dieses Feature angeboten.

Durch die obige Implementierung werden die Ergebnisse dynamisch errechnet und stellen keinen zusätzlichen Speicherverbrauch dar.

6.2.3.2 Importservice

Ebenso wie der Berechnungsservice wurde auch die Implementierung eines Importservice, in der Anforderungsliste der autosomalen Datenbank (Kapitel 5.1.1) verankert. Mit ihm hat der Anwender die Möglichkeit die Webanwendung mit Daten anzureichern. In Abbildung 6-12 sind die notwendigen Schritte, mittels eines Programmablaufplanes, die der Anwender beim Import eines Datensatzes durchläuft, beschrieben. Diese werden im Nachhinein partiell beschrieben.

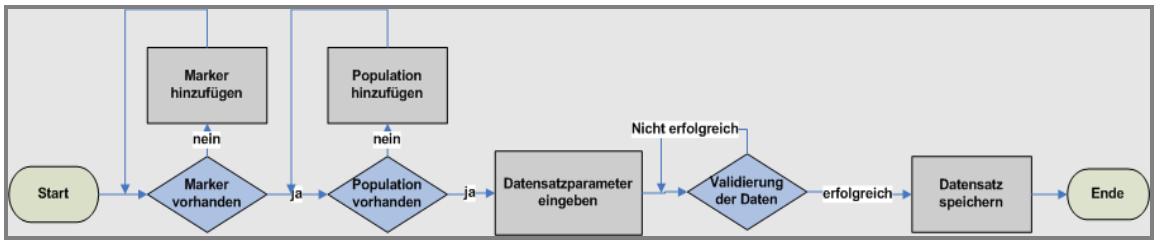


Abbildung 6-12: Flussdiagramm Daten-Import – beschreibt die notwendigen Schritte die der Nutzer beim Import von Daten durchläuft

In den ersten zwei Schritten muss der Anwender sich vergewissern, ob der *Marker* und die *Population* für die Daten vorliegen und bereits in der Datenbank erfasst wurden. Ist beides nicht der Fall, muss er beides über die bereitgestellten Formulare (Marker hinzufügen und Population hinzufügen) separat einpflegen. Sind diese zwei Entscheidungen erfüllt, muss er alle weiteren Parameter, die der Datensatz erfordert (Herausgeber, Autor, getestete Individuen, *Allelfrequenzen*, etc.) eingeben. Sind auch diese Kriterien erfüllt werden die Daten zuerst auf Gültigkeit überprüft. Zu dieser Validierung der Daten gehören u.a. eine Summierung der *Allelfrequenzen* mit einem Toleranzbereich und die Einhaltung der E-Mail-Syntax. Nach erfolgreicher Kontrolle der Daten werden sie in der Datenbank gespeichert.

Dieser angebotene Service wurde mit Hilfe einer *Session Bean* (Kapitel 3.4) realisiert. Dies bot sich an, da die *Session Bean*, mit Hilfe der *Annotations* unter *Seam*, direkt in einer Anzeigeseite aufgerufen und verwendet werden kann (Kapitel 4.2.3). Der Quelltext in Abbildung 6-13 zeigt das Grundgerüst der *Session Bean* mit den wichtigsten Methoden und Attributen. Die *Getter-* und *Setter-Methoden*, sowie einige andere Attribute der *Session Bean*, die für die Erklärung des Dienstes unrelevant sind, wurden zur besseren Visualisierung des Quelltextes ausgeblendet.

```
1.  @Stateless
2.  @Name("submitdataaction")
3.  @Scope(ScopeType.SESSION)
4.  public class SubmitDataAction implements
5.  Serializable,SubmitDataActionImpl{
6.
7.  @PersistenceContext
8.  private EntityManager em;
9.
10.  @In
11. FacesMessages facesMessages;
12.
13. public void initialize() {}
14. public void addMarker() {}
15. public void addPopulation() {}
16. public void createFields() {}
17. public void addData() {}
18. }
```

Abbildung 6-13: Quellcodeausschnitt aus SubmitDataAction.java – demonstriert den Aufbau der Session Bean, welche den Importservice realisiert

In Zeile eins sieht man die *Annotation* @Stateless. Diese kennzeichnet die *JavaBean* als eine zustandslose Komponente. Anders wie z.B. beim Einkauf eines Kunden im Internet mit einem virtuellen Warenkorb (Speicherung der Produkte über die gesamte *Session*), der durch eine zustandsbehaftete *Session Bean* realisiert werden könnte, kann hier beim Import der Daten mit einer zustandslosen *Session Bean* gearbeitet werden. Der Benutzer gibt die Parameter des Datensatzes ein und speichert diese direkt im Anschluss, ohne nochmals andere Seiten aufzurufen. Es ist also eine Implementierung als zustandslose *Session Bean* aus Sicht des Autors ausreichend. Mit der Notation @Name in Zeile zwei wird ein Name für die *Session Bean* definiert, mit der sie im *Seam*-Kontext registriert werden soll. Zeile drei deklariert einen Standardgültigkeitsbereich für die *Session Bean*. In unserem Fall gilt dieser für die gesamte *Session* in der der Benutzer mit der Anwendung interagiert. In Zeile acht und elf werden zwei Klassen-Attribute definiert. Zum einen das Attribut em vom Typ *Entity Manager*, er verwaltet die Zuordnung (mapping) zwischen relationalen Datenbanktabellen und *Entity Bean*-Objekten. Das zweite Klassen-Attribut facesMessages, vom Typ *FacesMessages* ist für die Ausgabe von Meldungen auf den Anzeigeseiten zuständig. So werden dem Benutzer hilfreiche Informationen (z.B. Datensatz zurückgesetzt oder Datensatz gespeichert) über die Präsentationsschicht angezeigt. Zeile dreizehn bis siebzehn zeigt die Klassen-Methoden der *Session Bean*. Diese werden der *Session Bean* durch das implementierte *Interface* SubmitDataActionImpl vererbt. Diese einzelnen Klassen-Methoden werden nun im Folgenden stichpunktartig beschrieben.

- **initialize-Methode:** setzt alle Felder und Attribute beim Aufruf der *Session Bean* zurück
- **addMarker-Methode:** speichert ein nicht erfasstes *Marker*-Objekt in der Datenbank
- **addPopulation-Methode:** speichert ein nicht erfasstes *Population*-Objekt in der Datenbank
- **createFields-Methode:** instanziert zwei Arrays (*Allelfrequenzen* und *Allele*), die Größe des Arrays wird in Abhängigkeit von der Anzahl der Eingabefelder, die der Benutzer ausgewählt hat, bestimmt
- **addData-Methode:** speichert den Datensatz durch den *Entity Manager*

In Abbildung 6-14 sieht man das Import-Formular wie es im Web-Browser angezeigt wird. Ganz oben im Formular ist die Textausgabe Daten zurückgesetzt zu sehen. Diese Ausgabe wurde durch das Klassenattribut facesMessages erzeugt und auf der *JSF*-Seite zur Anzeige gebracht. Direkt darunter sind verschiedene Datensatzparameter zu sehen, die der Benutzer beim Import von Daten eintragen muss (z.B. Herausgeber, Autor und Referenz der Daten). Im mittleren Bereich kann der Benutzer die *Population* und den *Marker* über eine *SelectOneMenu-Komponente* aus der *JSF-Komponentenbibliothek* auswählen. Sind die gewünschten Einträge nicht vorhanden, muss er auf die blauen Links Population hinzufügen oder Marker hinzufügen klicken und gelangt somit zu einem neuen Formular, wo er diese einpflegen kann. Nach erfolgreicher Eingabe werden diese dann mit in der *SelectOneMenu-Komponente* angezeigt.

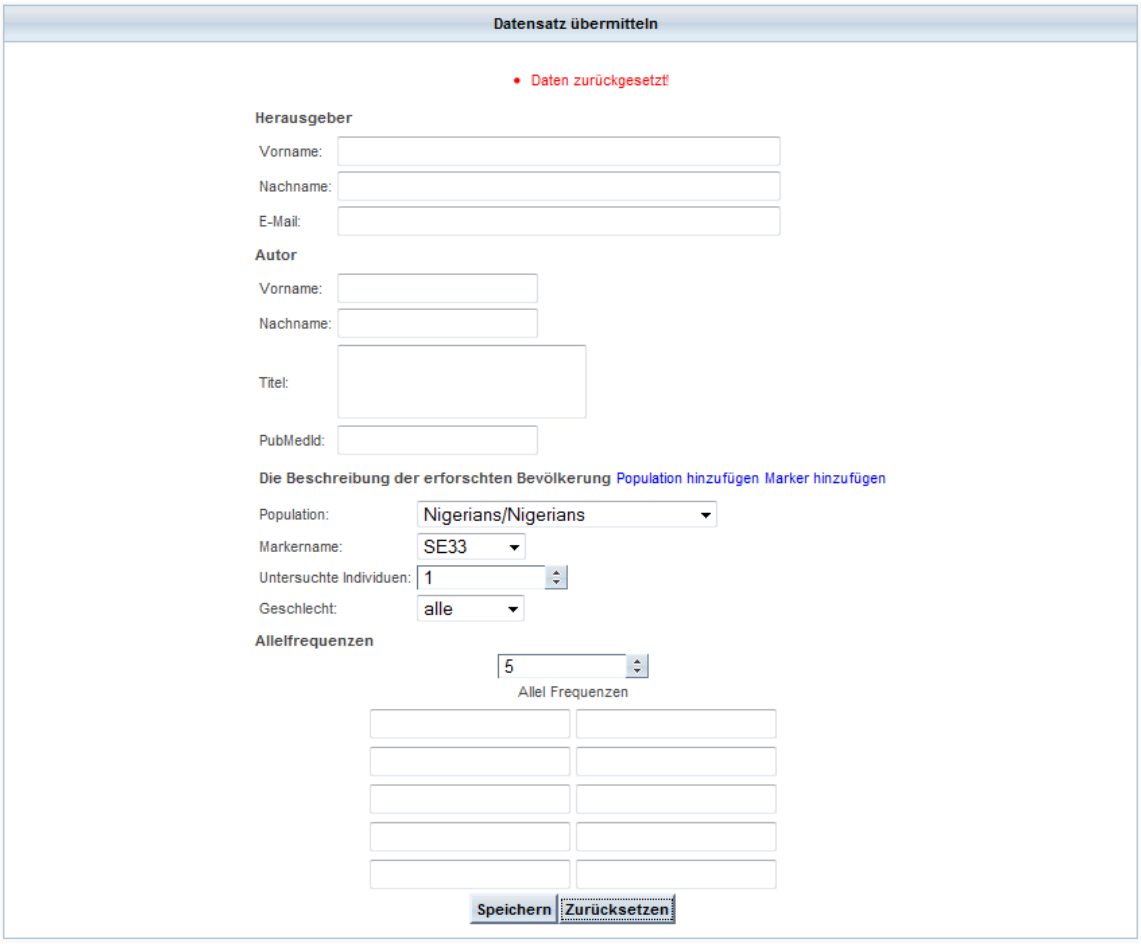


Abbildung 6-14: Importformular

Im unteren Bereich des Formulars kann der Anwender die Anzahl (in Abbildung 6-14 zwei mal fünf Eingabefelder) der Eingabefelder für die *Allele* und deren dazugehörige Frequenzwerte (siehe Kapitel 6.2.3.2 createFields-Methode) über eine *InputNumberSpinner-Komponente* aus der *RichFaces-Komponentenbibliothek* festlegen. Letztendlich kann er nach Eingabe aller optionalen Parameter, den Datensatz über den Button Speichern in der Datenbank speichern oder über den Button Zurücksetzen das Formular bei Fehleingaben zurücksetzen.

Fazit

Mit der Implementierung des Importservice ergab sich eine zum Teil aufwendige Möglichkeit Daten in die Datenbank zu speisen. Um jedoch einen funktionellen Kreislauf (Import von Daten → Export von Daten) zu gewährleisten, wurde dieser Dienst implementiert. Alternative Möglichkeiten für den Import von Daten sind empfehlenswert und werden im Kapitel 8 diskutiert.

6.2.3.3 Visuelles Feature (Weltkarte)

Als visuelles Feature wurde auf mehreren Unterseiten der autosomalen Datenbank eine Weltkarte integriert. Dieser Dienst wird durch die Firma Google Inc. bereitgestellt und kann an die individuellen Bedürfnisse beliebig angepasst werden. [61] [62]
Dieser Dienst stellt die geografische Lage von den erfassten *Populationen* der Datenbank mit Hilfe von Positionsmarkern grafisch dar. Der Anwender kann zwischen einer Kartendarstellung, einem Luftbild und einer Ansicht wählen, die sowohl eine Karte als auch das Luftbild darstellt. Durch verschiedene Navigationselemente, wie zum Beispiel einer Zoomfunktion, kann der Anwender sich im Karten- bzw. Bildausschnitt fortbewegen. Die Voraussetzungen zur Einbindung dieses Dienstes waren, das Einrichten eines Google Kontos und im Anschluss daran das Anfordern eines *API-Keys*. Der *API-Key* bindet die jeweilige Domain der Webanwendung an den Dienst von Google. Danach kann der Dienst individuell in die Webanwendung eingebunden und für die benutzerspezifischen Anforderungen angepasst werden. Die *Komponentenbibliothek JBoss RichFaces*, welche in der autosomalen Datenbank Einsatz findet, stellt für die Unterstützung des Dienstes eine Komponente bereit. In Abbildung 6-15 ist die Komponente mit dem dazugehörigen *API-Key* zu sehen. Mit dem unten gezeigten Key wurde während der Entwicklung der autosomalen Datenbank gearbeitet. Er gilt nur für die Domain http://localhost:8080/ADB/. Wenn die Webanwendung in Zukunft ins Netz gestellt wird, muss ein neuer Key angefordert werden, da sich der Name der Domain somit ändert (z.B. http://www.adb.qualitype.de).

```
1. <rich:gmap gmapKey="ABQIAAAA0sZkIkRZYnM6hSuvlvKMRRSOVA3VU2Nm01
2.  UmNykwo6cL31ZvIxQBBYq8O4cokrSoXrDjfmnrfto7g"/>
```

Abbildung 6-15: GoogleMaps-RichFaces-Komponente - demonstriert den Einsatz der Map-Komponente von RichFaces in einer JSF-Seite mit der Angabe des API-Keys

Der Dienst wurde in der autosomalen Datenbank auf der Markerseite, der Populationsseite, sowie der Mapseite (zeigt alle erfassten *Populationen* der Datenbank) implementiert. Im Nachfolgenden wird die Implementierung auf der Mapseite erläutert.
Im ersten Schritt wurde die obige *RichFaces*-Komponente auf der Anzeigeseite showMap.xhtml eingebunden (Abbildung 6-16 Zeile eins bis Zeile drei). Die Komponente besitzt außer dem *API-Key* noch weitere Attribute. Zum einen wird ihr der Variablenname map und eine feste Breite von 665 Pixeln zugeteilt. Des Weiteren wird bei der Initialisierung der Komponente eine Funktion namens initialize aufgerufen.

```
1. <rich:gmap  oninit="initialize(map)" gmapKey="
2.  ABQIAAAA0sZkIkRZYnM6hSuvlvKMRRSOVA3VU2Nm01UmNykwo6cL31ZvIx QBBY-
3.  q8O4cokrSoXrDjfmnrfto7g"  style="width:665px"  id="map"/>
4.  <a4j:repeat var="p" value="#{populationsOnMap}">
5.    <script type="text/javascript">
6.          addmarker(
7.                 '<h:outputText value="#{p.longitude}"/>',
8.                 '<h:outputText value="#{p.latitude}"/>',
9.                 '<h:outputText value="#{p.name}"/>',
10.                '<h:outputText value="#{p.id}"/>',
11.                '<h:outputText value="#{p.continent}"/>',
12.                '<h:outputText value="#{p.country}"/>',
13.                '<h:outputText value="#{p.ethnicSpecification}"/>'
14.       );
15.    </script>
16.  </a4j:repeat>
```

Abbildung 6-16: Quellcodeausschnitt aus showMap.xhtml – demonstriert wie das Array (populationsOnMap) mit den einzelnen Populationen durchlaufen wird und für jede Population ein Marker auf der Weltkarte gesetzt wird

Diese Funktion definiert das Aussehen (z.B. Kartenformat, Zoom) der Weltkarte beim erstmaligen Zugriff auf sie. In Zeile vier wird das Array populationsOnMap aus der *Session Bean* MapAction durchlaufen. Diese *Session Bean* lädt alle *Populationen* aus der Datenbank, speichert sie in einem Array und stellt sie der jeweiligen Anzeigeseite zur Verfügung. Abbildung 6-17 zeigt den Quelltext der *Session Bean*. In Zeile zehn wird die Variable populationsOnMap vom Typ List<PopulationEntity>, welche auf der *JSF*-Seite aufgerufen und durchlaufen wird, definiert. In Zeile fünfzehn und sechzehn der *Session Bean* wird diese Variable mit Hilfe des *Entity Managers* mit den *Populationen* befüllt.

```
1. @Stateless
2. @Name("mapaction")
3. @Scope(ScopeType.SESSION)
4. public class MapAction implements MapActionImpl,Serializable{
5.
6.    @PersistenceContext
7.    EntityManager em;
8.
9.    @Out (required=false) @In (required=false)
10.   public List<PopulationEntity> populationsOnMap;
11.
12.   @SuppressWarnings("unchecked")
13.   public void findPopulations(){
14.
15.         populationsOnMap = em.createQuery("from Population p")
16.                                 .getResultList();
17.   }
18.}
```

Abbildung 6-17: Quellcodeausschnitt aus MapAction.java – demonstriert den Aufbau der Session Bean, welche den Mapservice implementiert (zieht alle Populationen aus der Datenbank und speichert sie in einem Array)

Wie in Abbildung 6-16 zu sehen war, wurde nach der Einbindung der Kartenkomponente das Array populationsOnMap mit den *Populationen* iterativ mit der R*epeat-Komponente* durchlaufen. Bei jedem Durchlauf wird die Methode addMarker aufgerufen. Diese Methode bekommt als Parameter die Attribute (Längengrad, Breitengrad, Name und Kontinent der *Population*) des aktuellen *Population*-Objektes des Array mit den *Populationen* übergeben. Die aktuelle *Population* wird daraufhin auf der Weltkarte in Form eines Markers positioniert. Für den Anwender ist diese schrittweise Positionierung nicht ersichtlich. Er bekommt als Resultat alle *Populationen* auf einen Blick, wie in Abbildung 6-18 zu sehen ist.

Abbildung 6-18: Weltkartenservice

Fazit

Was die Implementierung der Weltkarte betrifft, so konnte anhand dieser Funktionalität gezeigt werden, dass sich dieser Service als besonders nutzerfreundlich auszeichnet. Der Anwender bekommt somit einen schnellen Überblick über die erfassten *Populationen* und kann diese schnell und komfortabel ansteuern.

6.2.3.4 Internationalisierung

Mit Hilfe des *Seam* Frameworks bot sich ein einfacher Mechanismus für die Realisierung einer mehrsprachigen Webanwendung an. Dabei wurden keine

sprachspezifischen Texte im Quelltext eingebettet. Mittels einer bestimmten Hilfsklasse können sprachspezifische Texte in Java-*Properties-Dateien* ausgelagert werden. [33] Die Klasse bietet eine Funktion zum Erkennen von Änderungen, die zur Laufzeit in der *Properties-Datei* gemacht werden, an. Diese Änderungen werden an die Applikation weitergeleitet.

Für den Prototyp wurden zwei *Properties-Dateien* (deutsch und englisch) angelegt. Abbildungen 6-19 und 6-20 zeigen Ausschnitte aus den zwei *Properties-Dateien* für deutschsprachige und englischsprachige Spracheinstellungen.

```
1. output.menu.first.header=Start
2. output.menu.first.one=Über uns
3. output.menu.first.two=Erläuterungen
4. output.menu.first.three=Weltkarte
```

Abbildung 6-19: Quellcodeausschnitt aus messages_de.properties – demonstriert den Aufbau einer Properties-Datei (deutsche Spracheinstellung)

```
1. output.menu.first.header=Home
2. output.menu.first.one=About us
3. output.menu.first.two=Introduction
4. output.menu.first.three=World map
```

Abbildung 6-20: Quellcodeausschnitt aus messages_en.properties - demonstriert den Aufbau einer Properties-Datei (englische Spracheinstellung)

Diese *Properties-Dateien* werden durch spezielle Textdateien repräsentiert. Sie besitzen einen bestimmten Aufbau und einen speziellen Namen. Ziel ist es die konkreten Textinformationen aus den Anzeigeseiten auszulagern und zentral zu verwalten. Ein einzelner Eintrag aus der *Properties-Datei* besteht immer aus einem *Message-Schlüssel* und dem dazugehörigen Wert. Der Zugriff auf die jeweilige Textinformation geschieht durch Angabe des jeweiligen Schlüssels in der Anzeigeseite (siehe Abbildung 6-21).

```
1. <h:outputText value="#{messages['output.menu.first.header']}"/>
```

Abbildung 6-21: Quellcodeausschnitt aus menu.xhtml – demonstriert den Zugriff auf eine Textinformation mittels Message-Schlüssel

Die Sprachauswahl wurde im Kopfbereich der Webanwendung implementiert. Mit Hilfe des in Abbildung 6-22 gezeigten Sprachauswahlmenüs kann der Anwender zwischen zwei Sprachen (deutsch und englisch) auswählen.

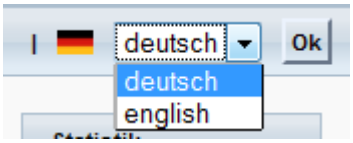


Abbildung 6-22: Sprachauswahl

Fazit

Zwar muss festgestellt werden, dass die zusätzliche Internationalisierungsfunktion nur mit großem Entwicklungsaufwand zu bewerkstelligen war, jedoch kann man im Hinblick auf die Zukunft erwarten, dass durch deren Implementierung ein Grundsystem vorliegt, welches einfach und schnell durch das Hinzufügen weiterer *Properties-Dateien*, die autosomale Datenbank um neue Sprachpakete erweitern kann.

6.2.3.5 Schnellsuche

Ebenfalls wie die Sprachauswahl wurde im Kopfbereich der Seite auch eine Schnellsuche eingebunden. Sie bietet dem Anwender die Möglichkeit schnell und komfortabel nach erfassten *Markern* in der autosomalen Datenbank zu suchen und zu diesen zu navigieren.

Diese Funktionalität wurde mit der speziell dafür vorgesehenen *SuggestionBox-Komponente* aus der *RichFaces-Tag Library* realisiert. Dies ist ein Eingabefeld, welches dem Anwender in Abhängigkeit von dem bereits eingegeben Wert Vorschläge zur Vervollständigung der Eingabe unterbreitet (siehe Abbildung 6-23).

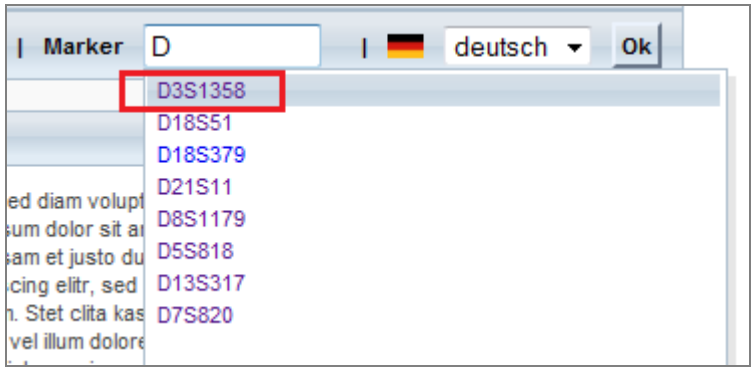


Abbildung 6-23: Schnellsuche – für die schnelle Suche nach Markern

Die *SuggestionBox-Komponente* wird über das Attribut for mit einem Eingabefeld und dessen Bezeichner (ID) verknüpft. Im Hintergrund wird die *SuggestionBox-Komponente* dann über die im Attribut suggestionAction angegebene Bean-Methode gesteuert (siehe Abbildung 6-24). Die Methode muss einen Input-Parameter vom Typ Object

akzeptieren. Dieser wird durch die Eingabe des Benutzers repräsentiert. Daraufhin liefert die Methode ein Rückgabewert vom Typ List<String>. Dieser Rückgabewert zeigt sich im Web-Browser durch die vorgeschlagenen Ergebnisse.

```
1. public List<String> showMarkerList(Object input) {
2.
3.              List<String> entriesFound = new ArrayList<String>();
4.              entriesFound = this.findMarkerAndSynonyms();
5.
6.              String userInput =  (String)input;
7.              ArrayList<String> ret = new ArrayList<String>();
8.
9.              for (String mark : entriesFound ) {
10.   if ((mark.toLowerCase()).startsWith(userInput.toLowerCase())) {
11.                   ret.add(mark);
12.                }
13.             }
14.             return ret;
15.         }
```

Abbildung 6-24: Quellcodeausschnitt aus MarkerAction.java - demonstriert den Aufbau der Methode showMarkerList, welche dem Benutzer in Abhängigkeit von der Eingabe einen Wert vorschlägt

Durch einen Klick auf einen vorgeschlagenen Wert, gelangt der Anwender dann auf die Seite des *Markers*. Bei der Eingabe des *Markers* in der *SuggestionBox-Komponente* wurden auch Synonyme berücksichtigt. Gibt der Anwender ein Synonym zu einem *Marker* ein, gelangt er ebenfalls auf die Seite des jeweiligen *Markers*.

Fazit

Mit der Implementierung der Schnellsuche ergab sich eine Vereinfachung für den Anwender, um Markerseiten einfach und komfortabel aufzurufen. Zwar kann gesagt werden, dass die *SuggestionBox-Komponente* nicht auf jedes Layout einer Webanwendung zugeschnitten werden kann, doch ist es für die autosomale Datenbank, welche ohne stark eingegrenzte Layoutrichtlinien seitens der Firma Qualitype AG auskommt, ein optimaler Lösungsansatz für eine Implementierung einer Schnellsuch-Funktionalität.

6.2.3.6 Exportservice

Um Daten aus der Webanwendung exportieren zu können, wurde ein Exportservice implementiert. Dieser Service wurde mit Hilfe der *JExcel-API* in einer Java-Klasse realisiert. [63]

JExcel kann alle Excel-Dateien von der Version 1997 bis 2003 interpretieren und Dateien erzeugen, die zu diesen Versionen kompatibel sind. [64]

Für die Implementierung des Dienstes müssen die benötigten *JAR-Dateien* der *JExcel-API* in ein Java-Projekt importiert werden. Danach kann die Funktionalität der *API* in vollem Umfang genutzt werden. Der in Abbildung 6-25 gezeigte Quelltext zeigt den Grundaufbau der Java-Klasse mit den dazugehörigen Klassen-Attributen und Klassen-Methoden. Die Methoden werden im Folgenden stichpunktartig erläutert.

```
1. public class Export {
2.
3.    private WritableWorkbook workbook;
4.    private WritableSheet testSheet;
5.    private int currentColumn;
6.    private int currentRow;
7.
8.    Export () {}
9.
10.   public void createFile(String fileName) {}
11.   public void closeFile(){}
12.   public void writeDate(List<Populationset> setList,AlleleName[]
13.   AlleleArray) {}
14    public void createSheet() {}
15. }
```

Abbildung 6-25: Quellcodeausschnitt aus Export.java – demonstriert den Aufbau der Java-Klasse, welche für den Export von Daten verantwortlich ist

- **createFile:** instanziert eine neue *xls-Datei* ohne Inhalt
- **closeFile:** schließt alle Referenzen auf die *xls-Datei* und speichert sie lokal auf dem Datenträger
- **writeDate:** schreibt die jeweiligen Datensätze in die *xls-Datei*
- **createSheet:** erstellt ein neues Arbeitsblatt für die *xls-Datei*

Der Exportservice kann bisher im Prototyp nur auf der Markerseite verwendet werden. Ein Einsatz auf weiteren Unterseiten der Webanwendung soll in Kapitel 8 diskutiert werden. Möchte der Benutzer z.B. Datensätze von der Markerseite exportieren, muss er über eine *Checkbox-Komponente*, des dazugehörigen Datensatzes, den gewünschten Datensatz selektieren und anschließend auf den Button Export klicken (siehe Abbildung 6-26).

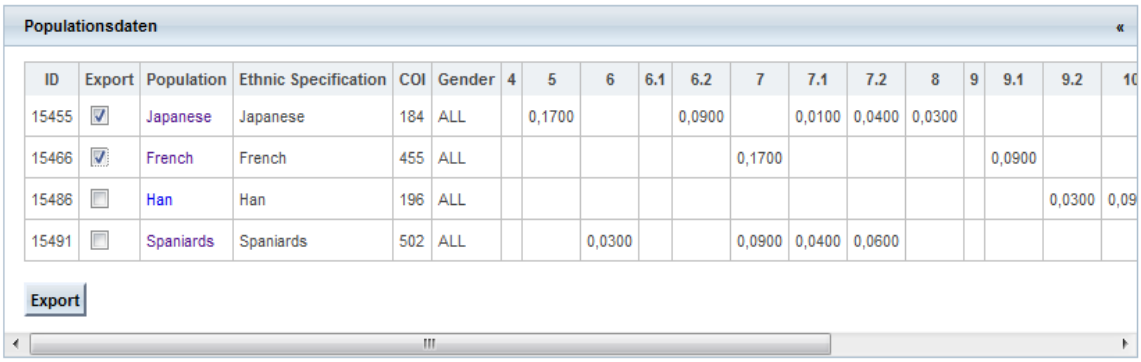


Abbildung 6-26: Exportservice - die selektierten Datensätze Japan und French werden nach dem Klick auf den Export-Button in einer xls-Datei gespeichert

Als Ergebnis erhält der Anwender dann eine *xls-Datei*, die er mit Microsoft Excel öffnen und sich anzeigen lassen kann.

Fazit

Ziel der Implementierung des Exportservice war es, dem Nutzer eine Möglichkeit zu bieten, Daten nicht nur visuell auf der Website betrachten zu können, sondern auch für weitere Verwendungszwecke auf sein System herunterladen zu können. Dabei ergab es sich, dass die Implementierung des Service für weitere Unterseiten (Populationsseite) zwingend erforderlich ist und im Kapitel 8 unter Ausblicke diskutiert wird.

6.3 Test der Webanwendung

Ein wichtiger Bestandteil während der Softwareentwicklung ist das Testen. Bei der Entwicklung der autosomalen Datenbank wurden während der Programmierungsphase Komponententests mit dem Testframework *TestNG* durchgeführt. [65] Weiterhin wurden Tests zum Verhalten der Anwendung in verschiedenen Web-Browsern durchgeführt. Im nachfolgenden Abschnitt werden Informationen zu den unterschiedlichen Tests erläutert.

6.3.1 Komponententest

Beim Komponententest werden die einzelnen Module (Units, Methoden und Funktionen) der Anwendungslogik atomar und unabhängig von einer Umgebung und Abhängigkeit getestet. Ziel dieser Tests ist es, bereits frühzeitige, während der Entwicklungsphase auftretende Programmfehler, in den Komponenten der Software zu

lokalisieren und zu beheben. Die Funktionalität der Module kann so einfacher getestet werden, als wenn die Module bereits integriert sind, da in diesem Fall die Abhängigkeiten zwischen den Modulen in Betracht gezogen werden müssen. Als Werkzeug für den Komponententest, wurde wie bereits oben erwähnt, das Open Source Framework *TestNG* genutzt. Die spezifischen Testfälle sind selbst in der Programmiersprache Java codierte Klassen. *TestNG* konnte ohne große Probleme nach [66] in die Entwicklungsumgebung *Eclipse* integriert werden. Abbildung 6-27 zeigt einen Ausschnitt der Testklasse CalculationTest. Mit dieser Java-Klasse wurden die erwarteten biostatistischen Berechnungsparameter getestet.

```
1.  @Test
2.  public void testHomozygotie() {
3.  Assert.assertEquals(calc.getH(allels).setScale(4,1).compareTo(new
4.     BigDecimal(0.2154).setScale(4,1)),  0 );
5.    }
```

Abbildung 6-27: Quellcodeausschnitt aus CalculationTest – demonstriert den Aufbau einer Test-Methode mit TestNG als Framework

Nach der Durchführung des Tests bekommt der Entwickler eine Auswertung wie in Abbildung 6-28 dargestellt. Bei der Auswertung gibt es nur zwei Zustände: Test bestanden oder Test nicht bestanden. An dem grünen Haken in Abbildung 6-28 (testHomozygotie()) ist zu erkennen, dass der berechnete Erwartungswert von 0.2154 mit dem errechneten Wert der Methode übereinstimmt, also der Test bestanden ist. Um diesen Test als repräsentative Aussage nutzen zu können, wurde dieses Szenario für verschiedene *Allelfrequenzen* durchgeführt.

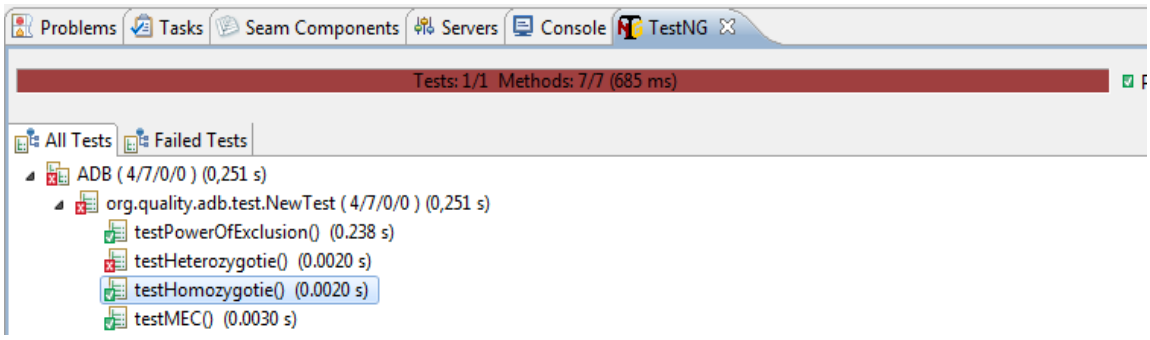


Abbildung 6-28: TestNG Auswertung – Test von Methoden mit Test-Framework TestNG, drei Tests (testPowerOfEcclusion, testHomozygotie, testMEC) erfolgreich (grüner Haken), ein Test (testHeterotygothie) nicht erfolgreich (rotes Kreuz)

6.3.2 Browsertest

Da die Web-Browser von verschiedenen Herstellern unterschiedlich reagieren, wurde ein Browser-Test durchgeführt. Um den Aufwand an Browser-Tests zu reduzieren, wurde die Anzahl auf drei Web-Browser festgelegt. Ausgewählt wurden die zwei meistgenutzten Web-Browser Internet Explorer und Firefox (siehe Tabelle 6-1). Um die Fehleranfälligkeit auch in weniger gebräuchlichen Web-Browsern zu ermitteln, wurde der Web-Browser Opera ausgewählt, weil dieser nur gering verbreitet war. Beim Web-Browser-Test wurden folgende Versionen getestet. Beim Internet Explorer von Microsoft die Version 8, beim Mozilla Firefox die Version 3.0.6 und beim Opera die Version 10.61.

2010	Internet Explorer	Firefox	Chrome	Safari	Opera
September	31.1 %	45.1%	17.3%	3.7%	2.2%
August	30.7 %	45.8%	17.0%	3.5%	2.3%
July	30.4 %	46.4%	16.7%	3.4%	2.3%
June	31.0 %	46.6%	15.9%	3.6%	2.1%
May	32.2 %	46.9%	14.5%	3.5%	2.2%
April	33.4 %	46.4%	13.6%	3.7%	2.2%
March	34.9 %	46.2%	12.3%	3.7%	2.2%
February	35.3 %	46.5%	11.6%	3.8%	2.1%
January	36.2 %	46.3%	10.8%	3.7%	2.2%

Tabelle 6-1: Verbreitung von Web-Browsern 2010 – zeigt die meist eingesetzten Web-Browser, drei wurden für den Test ausgewählt (rote Markierung) [67]

Mit Hilfe von Tabelle 6-2 wurden verschiedene Oberflächentests anhand der Kriterien aus [68, S. 145f] durchgeführt und ausgewertet. Diese Auswertung hat gezeigt, dass sich das Erscheinungsbild der Webanwendung im Auflösungsbereich 1920x1200 Pixel bis 1024x768 Pixel optimal verhält. Die verschiedenen Elemente der Anwendung zeigten in allen Web-Browsern die gleiche Darstellung. Weiterhin wurden alle *Cascading Style Sheets* (*CSS*) richtig interpretiert. Alle Navigationselemente der Webanwendung waren auffindbar und funktionstüchtig. Jedes Navigationselement besaß eine gültige Referenz zu einer vorhandenen Seite. Im Web-Browser Internet Explorer und Opera war als nachteilig zu bewerten, dass die Elemente im Formular „Datensatz übermitteln“, „Population hinzufügen“ und „Marker hinzufügen“ unkorrekt angezeigt wurden. Diese Fehlfunktion der Anzeige sollte im Nachhinein noch korrigiert werden. Die Druckfunktion erzielte in allen Browsern das gewünschte Ergebnis.

	Micorsoft Internet Explorer 8	Mozilla Firefox 3	Opera 10
Lesbarkeit&Handhabbarkeit bei Veränderung der Schriftart und Schriftgröße gewährleistet	Schriftart wird durch Seitenlayout definiert, **Test bestanden**	Schriftart wird durch Seitenlayout definiert, **Test bestanden**	Schriftart wird durch Seitenlayout definiert, **Test bestanden**
Haben alle gleichen Elemente die selbe Darstellung (z.B. Farbe, Größe von Buttons)	**Test bestanden**	**Test bestanden**	**Test bestanden**
Werden alle Elemente im Browser richtig dargestellt	Elemente auf den Importformularen werden nicht richtig dargestellt, **Test nicht bestanden**	**Test bestanden**	Elemente auf den Importformularen werden nicht richtig dargestellt, **Test nicht bestanden**
Werden die verwendeten Cascading Style Sheets richtig interpretiert	**Test bestanden**	**Test bestanden**	**Test bestanden**
Wird die Anwendung im Auflösungsbereich (1920x1200 px – 1024x768 px) richtig dargestellt	**Test bestanden**	**Test bestanden**	**Test bestanden**
Sind alle Navigationselemente auffindbar (Buttons, Hyperlinks)	**Test bestanden**	**Test bestanden**	**Test bestanden**
Funktionieren alle Navigationselemente (Buttons, Menüpunkte)	**Test bestanden**	**Test bestanden**	**Test bestanden**
Navigationsleiste auf jeder Seite sichtbar	**Test bestanden**	**Test bestanden**	**Test bestanden**
Arbeitet die Druckfunktion des Browsers korrekt	**Test bestanden**	**Test bestanden**	Bereich Statistik/ Neuigkeiten wird bei zu großer Schriftart versetzt gedruckt, **Test nicht bestanden**

Tabelle 6-2: Web-Browser Test – Kriterien (links), getestete Web-Browser (oben)

7 Anwendungsszenario

Dieses Kapitel zeigt zwei mögliche Anwendungsszenarien, die die möglichen Einsatzgebiete der neuen Datenbank aufzeigt. Die Vorgehensweise des Anwenders wird dabei schrittweise erläutert. Vorweg werden Grundlegende Informationen zum Import von Daten, welche aus einer Veröffentlichung stammen, anhand einer wissenschaftlichen Publikation erläutert. Die Anwendungsszenarien richten sich an die im Prototyp implementierten Funktionalitäten.

7.1 Voraussetzung für den Import von Daten

Heutzutage werden für die forensische Untersuchung (Aufklärung krimineller Handlungen) und Abstammungstests *Alleldiversitäten* unterschiedlicher *Populationen* der Weltbevölkerung verwendet. Für die Veröffentlichung dieser Untersuchungsergebnisse gab es jedoch kein einheitliches Format. Somit wurde im Jahr 2000 beschlossen, eine feste Methodik für die Bereitstellung von genetischen Populationsdaten einzuführen. Die Grundidee war es, die Veröffentlichung der Daten zu erleichtern und auf die Schlüsselinformationen für die wissenschaftliche Verwendung zu reduzieren. Ziel war es, ein kurzes, prägnantes Datenformat zu definieren, welches allgemein festgelegten Regeln folgt. [69] Durch diese Festlegung wurden auch Wissenschaftler in Entwicklungsländern angeregt, genetische Populationsdaten ihrer Bevölkerung zu publizieren. Der Aufbau und Inhalt dieser wissenschaftlichen Publikation ist folgendermaßen definiert:

- **Beschreibung der Population:** Die untersuchte *Population* sollte in variabler Größe beschrieben werden. Dabei ist darauf zu achten, dass auf die geografische Lage, Volkszugehörigkeit, Methoden der Untersuchung und Charakteristika zur *Population* eingegangen wird.
- **Ethnische Voraussetzungen:** In diesem Teil muss die Zustimmung der Untersuchten durch eine Kommission gewährleistet sein. Weiterhin sollten die Daten anonymisiert werden, da in manchen Ländern die Untersuchung spezieller *DNA*-Bereiche (Bestimmung des Geschlechts) verboten ist. Wichtig ist auch, dass die Autoren der Publikation die Regeln zur Veröffentlichung und die weitere Verwendung der Daten akzeptieren.

- **Qualitätskontrolle:** In diesem Bereich müssen die Autoren die Qualität der Daten gewährleisten. Die Verfahren der Qualitätssicherung müssen dabei ebenfalls genannt werden. Bei der Qualitätskontrolle ist stets den Regeln der *International Society for Forensic Genetics* (*ISFG*) zu folgen.
- **Allelfrequenzen/biostatistische Berechnungen:** In diesem Teil der Publikation werden die *Allelfrequenzen* (meist in einer Tabelle) angegeben. Einige Publikationen bieten zusätzlich auch Informationen zu biostatistischen Berechnungen und den sich daraus ergebenen Parametern an.

Diese Grundparameter aus der wissenschaftlichen Publikation wurden als Eingabemöglichkeiten im Importformular für die autosomale Datenbank eingearbeitet. Im Folgenden wird das erste Anwendungsszenario unter Verwendung einer wissenschaftlichen Publikation erläutert.

7.2 Anwendungsszenario 1

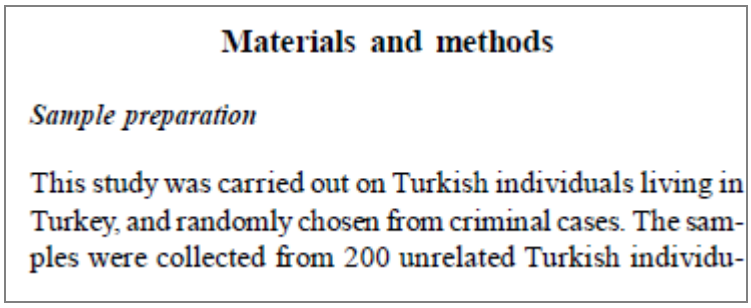
Problemstellung: Ein Forensiker möchte seine Daten mit Hilfe der autosomalen Datenbank für andere Wissenschaftler publizieren.

Daten: *Population* (200 Türken), *Allelfrequenzen* (*Marker* D3S1358 mit 8 *Allelen*)

Voraussetzungen: Web-Browser, Internetzugang

Schritt 1 – Daten für den Import definieren

Wie aus Kapitel 7.1 hervorging stellt eine wissenschaftliche Publikation die grundlegenden Parameter für den Datenimport zur Verfügung. Abbildung 7-1 zeigt den Textausschnitt der Publikation aus den Anlagen Teil 5. Aus diesem ist zu entnehmen, dass 200 unverwandte Türken an dieser Studie beteiligt waren. Tabelle 7-1 zeigt die dazugehörigen *Allelfrequenzen*. Die rot markierten Daten (*Marker* D3S1358 mit 8 *Allelen*) aus der Tabelle werden nun in den nächsten Schritten in die autosomale Datenbank eingepflegt.



Materials and methods

Sample preparation

This study was carried out on Turkish individuals living in Turkey, and randomly chosen from criminal cases. The samples were collected from 200 unrelated Turkish individu-

Abbildung 7-1: Textausschnitt aus einer wissenschaftlichen Publikation – Textausschnitt aus Paper aus den Anlagen Teil 5

Allele	D3S1358	vWA	FGA	D8S1179	21S11	D18S51	D5S818	D13S317	D7S820
7							0.002		0.005
8				0.015			0.007	0.130	0.162
9				0.015		0.002	0.052	0.070	0.090
10				0.062		0.007	0.107	0.082	0.250
11				0.050		0.020	0.340	0.305	0.257
12	0.005			0.110		0.097	0.320	0.275	0.162
13	0.002	0.002		0.280		0.147	0.162	0.100	0.050
13.2						0.020			
14	0.072	0.072		0.222		0.137	0.007	0.035	0.017
14.2						0.027			
15	0.250	0.122		0.147		0.125		0.002	0.005
16	0.337	0.195		0.072		0.157			
17	0.180	0.302		0.020		0.090			
18	0.137	0.202	0.005	0.005		0.065			
19	0.015	0.087	0.040			0.047			
20		0.015	0.090			0.030			
21			0.190			0.015			
22			0.140			0.010			
23			0.205						
24			0.170						
25			0.110						
26			0.040		0.002				
26.2									
27			0.007		0.015				
28			0.002		0.127				
29					0.222				
29.2					0.002				
30					0.235				
30.2					0.022				
31					0.055				
31.2					0.132				
32					0.012				
32.2					0.110				
33					0.002				
33.2					0.045				
34					0.002				
34.2					0.010				
35					0.002				

Tabelle 7-1: Frequenztabelle aus einer wissenschaftlichen Publikation - Allelfrequenzen von 9 STR-Markern einer türkischstämmigen Population, Ausschnitt aus einer wissenschaftlicher Publikation aus den Anlagen Teil 5

Schritt 2 – Webanwendung aufrufen

Als erstes muss der Forensiker die *URL* im Web-Browser (siehe Abbildung 7-2), unter der die Webanwendung verfügbar ist, aufrufen. Daraufhin wird er durch die Startseite der autosomalen Datenbank begrüßt und geführt.

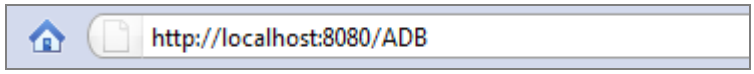


Abbildung 7-2: URL – Eingabe der URL im Web-Browser

Schritt 3 – Überprüfen, ob sich der Marker bereits in der Datenbank befindet

Zunächst sollte der Forensiker überprüfen, ob sich der *Marker* D3S1358 bereits in der Datenbank befindet. Dazu bietet sich die Markertabelle an. In ihr sind alle autosomalen *Marker* der Datenbank aufgelistet. Über das Navigationsmenü navigiert er auf dem

schnellsten Wege dorthin (Navigation: Datensätze → Markertabellen). In Abbildung 7-3 ist ersichtlich, dass sich der *Marker* bereits in der Datenbank befindet.

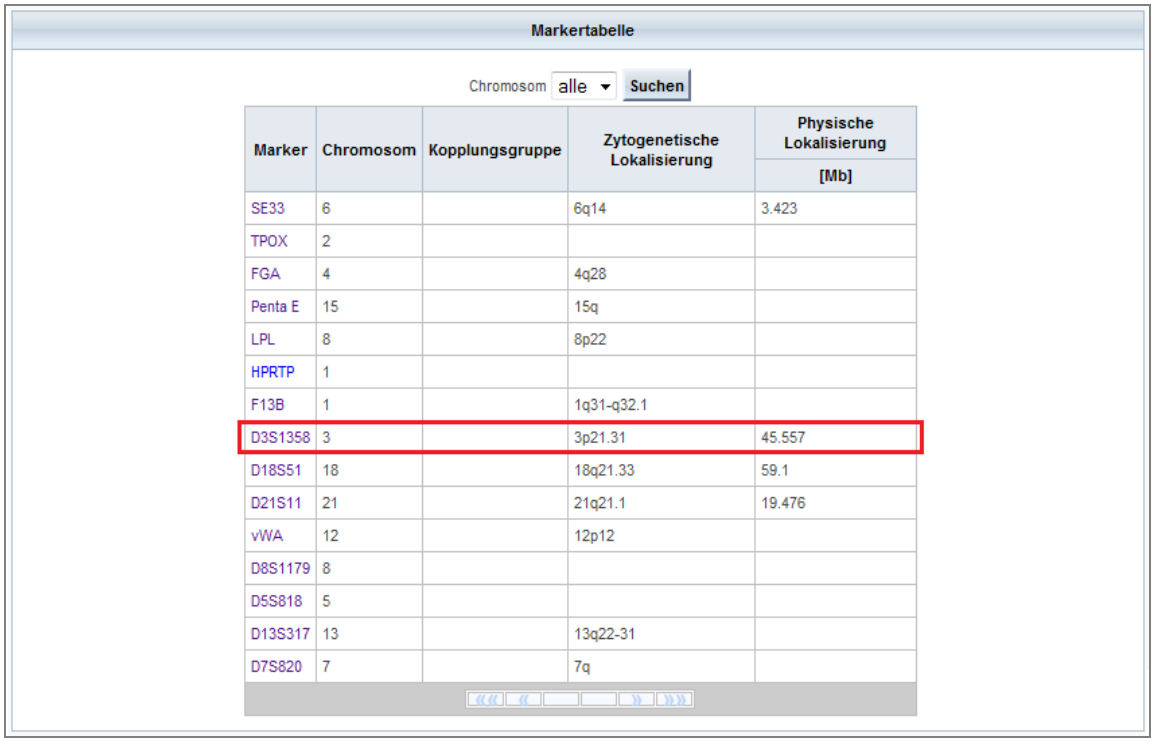


Abbildung 7-3: Markertabelle – Anzeige im Web-Browser, zeigt alle erfassten Marker der Datenbank mit ihren Attributen

Schritt 4 – Überprüfen, ob sich die Population bereits in der Datenbank befindet

Nachdem sichergestellt ist, dass sich der *Marker* bereits in der Datenbank befindet, sollte nach derselben Vorgehensweise auch überprüft werden, ob die *Population* Türkei bereits in der Datenbank vorhanden ist. Wieder bietet sich das Navigationsmenü als Werkzeug an. Über Datensätze → Populationstabellen kann er schnell zum gewünschten Punkt navigieren. Abbildung 7-4 zeigt, dass auch die *Population* Türkei bereits in der Datenbank erfasst wurde.

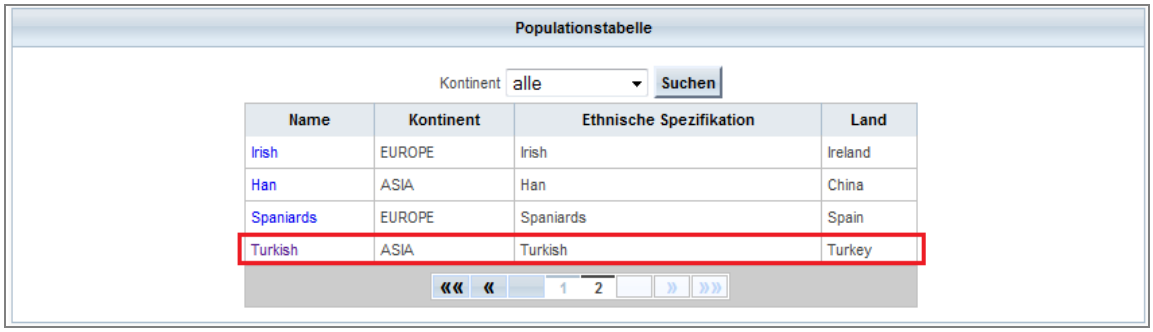


Abbildung 7-4: Populationstabelle – Anzeige im Web-Browser, zeigt alle erfassten Populationen der Datenbank mit ihren Attributen

Schritt 5 – Importformular aufrufen

Nach dem die vorherigen Schritte erfolgreich durchgeführt wurden, kann der Wissenschaftler nun zum Importformular navigieren. Dabei unterstützt ihn wiederum das Navigationsmenü. Über Datensätze → Datensatz übermitteln, ruft er das Importformular auf (siehe Abbildung 7-5).

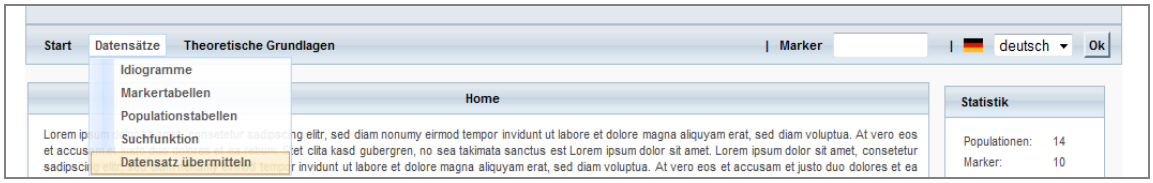


Abbildung 7-5: Navigationsmenü – Anzeige im Web-Browser, Navigation mit Hilfe des Navigationsmenüs zum Import-Formular

Schritt 6 – Datensatzparameter eingeben

In diesem Schritt werden alle Parameter mit Hilfe des Importformulars für den Datensatz eingegeben. Zuerst muss der Herausgeber (Vorname, Nachname und E-Mail) und Autor (Vorname und Nachname) der Daten benannt werden. In unserem Fall ist der Forensiker der Herausgeber und zugleich Autor der Daten. Danach kann eine Referenz zum Datensatz angeben werden (z.B. forensische Zeitschrift oder Internetseite der wissenschaftlichen Publikation). Anschließend muss die *Population* und der autosomale *Marker* über ein Auswahlmenü ausgewählt werden. Weiterhin ist anzugeben wie viele Individuen getestet wurden und um welches Geschlecht es sich dabei handelte. Zu Letzt müssen die 8 *Allele* mit den dazugehörigen *Allelfrequenzen* eingeben werden. Danach kann der Datensatz über den Button Speichern abgesendet werden. Abbildung 7-6 zeigt das ausgefüllte Formular mit den einzelnen Parametern.

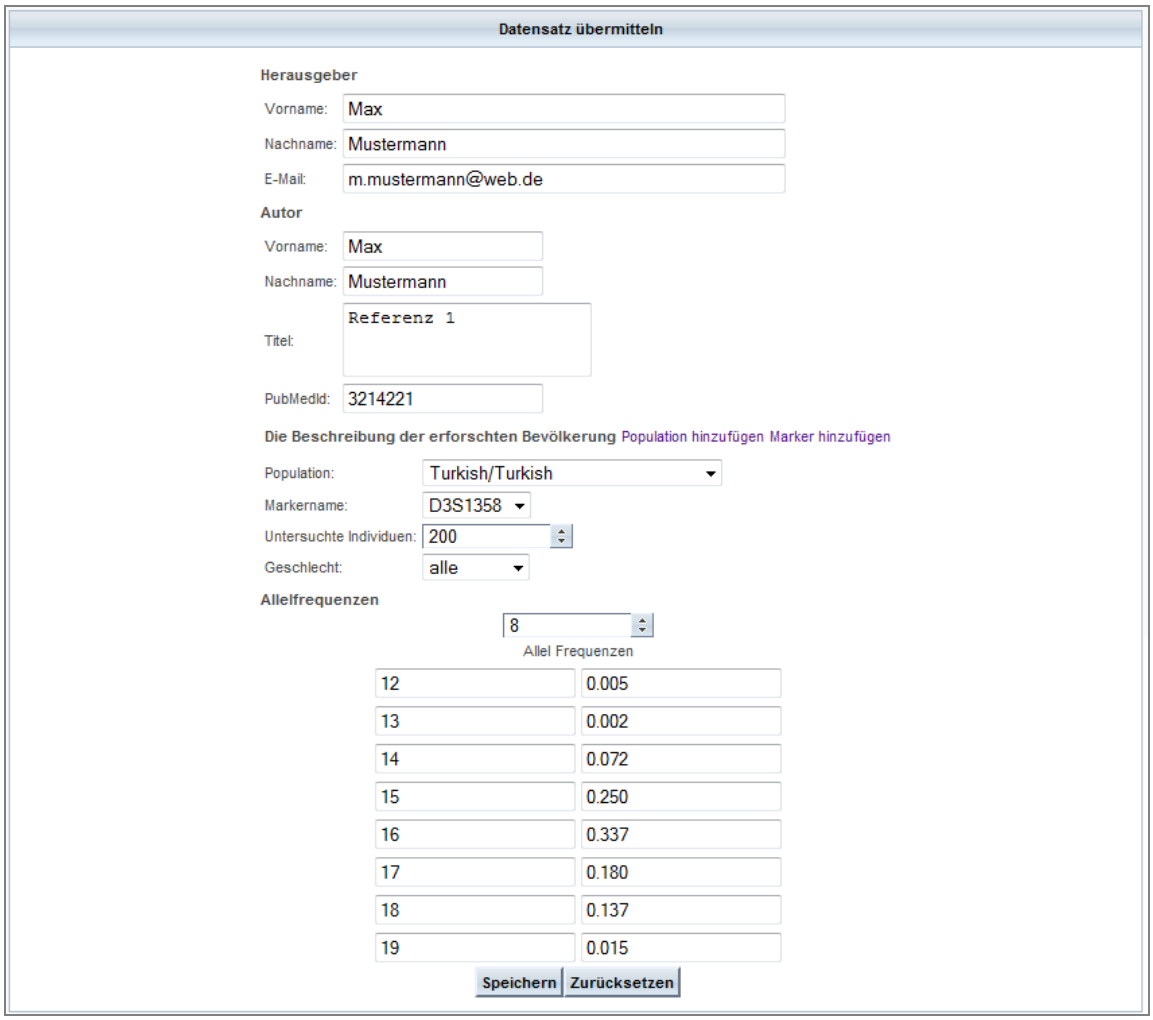


Abbildung 7-6: Import-Formular – Anzeige im Web-Browser, Eingabe der Parameter

In diesen sechs Schritten kann ein beliebiger forensischer Wissenschaftler einen Datensatz schnell und einfach weltweit publizieren.

7.3 Anwendungsszenario 2

Problemstellung: Ein Forensiker möchte bereits publizierte Daten des *Markers* D3S1358 mit Hilfe der autosomalen Datenbank für eigene Arbeiten verwenden und herunterladen.

Vorraussetzungen: Web-Browser, Internetzugang, Microsoft Office Excel

Schritt 1 – Webanwendung aufrufen

Schritt 1 in diesem Szenario ist äquivalent zum Schritt 2 aus Anwendungsszenario 1 und wird von daher nicht nochmals erläutert.

Schritt 2 – Marker in der Datenbank suchen

Zunächst muss der Forensiker den *Marker* D3S1358 suchen. Dazu bietet sich ihm die Schnellsuche im Kopfbereich der Webanwendung an. Wenn er im Eingabefeld den ersten Buchstabe des *Markers* eingibt, liefert ihm die Webanwendung alle *Marker*, die in der Datenbank erfasst sind und mit dem Buchstaben „D“ beginnen. In Abbildung 7-7 wird der gewünschte *Marker* ganz oben aufgelistet. Mit einem Klick auf den Namen kommt man direkt auf die Webseite des *Markers*.

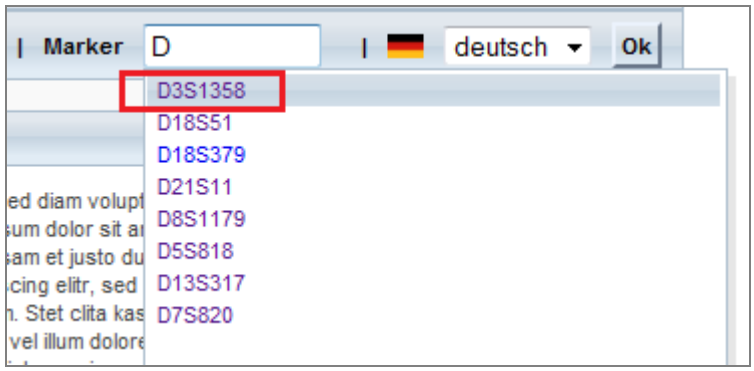


Abbildung 7-7: Schnellsuche 2 – Anzeige im Web-Browser, vorgeschlagene Ergebnisse der autosomalen Datenbank nach Eingabe des Buchstaben „D“

Schritt 3 – Publizierte Daten von der Markerseite herunterladen

Auf der Markerseite angekommen, liefert die Webanwendung dem Forensiker viele Informationen. Im oberen Bereich sind allgemeine Angaben zum *Marker* zu finden. Das *Panel* ist jedoch noch geschlossen. Möchte der Wissenschaftler es öffnen klickt er ganz einfach auf die Panelüberschrift Markerinformationen. Direkt darunter werden alle vorhanden *Populationen* mit den *Allelfrequenzen* zum *Marker* aufgelistet, soweit diese vorhanden sind. In Abbildung 7-8 sehen wir, dass die zwei *Populationen* Türkei und Venezuela zum *Marker* vorliegen. Weiterhin findet der Forensiker noch weitere *Panels* zum Öffnen (z.B. biostatistische Berechnungen, Referenzen und Weltkarte). Möchte der Forensiker diese Daten nun exportieren, muss er die gewünschten *Populationen* über die *CheckBox-Komponenten* selektieren und im Anschluss daran auf den Button Export klicken. Infolgedessen wird eine *xls-Datei* erzeugt (siehe Abbildung 7-9) und standardmäßig an einem Speicherort auf dem Rechner des Nutzers gespeichert.

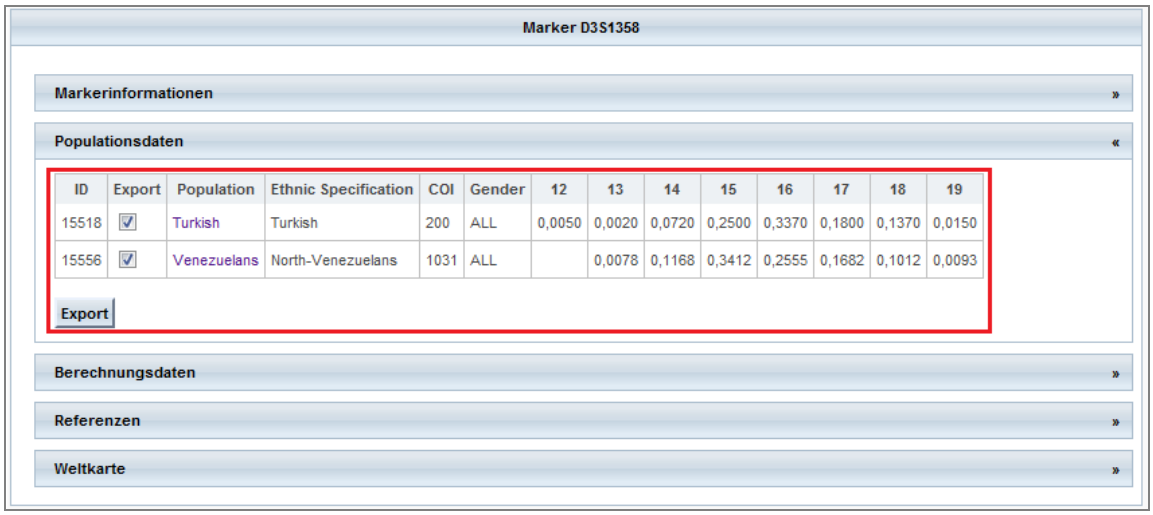


Abbildung 7-8: Markerseite – Anzeige im Web-Browser, Panel Populationsdaten geöffnet (beinhaltet zwei Populationen)

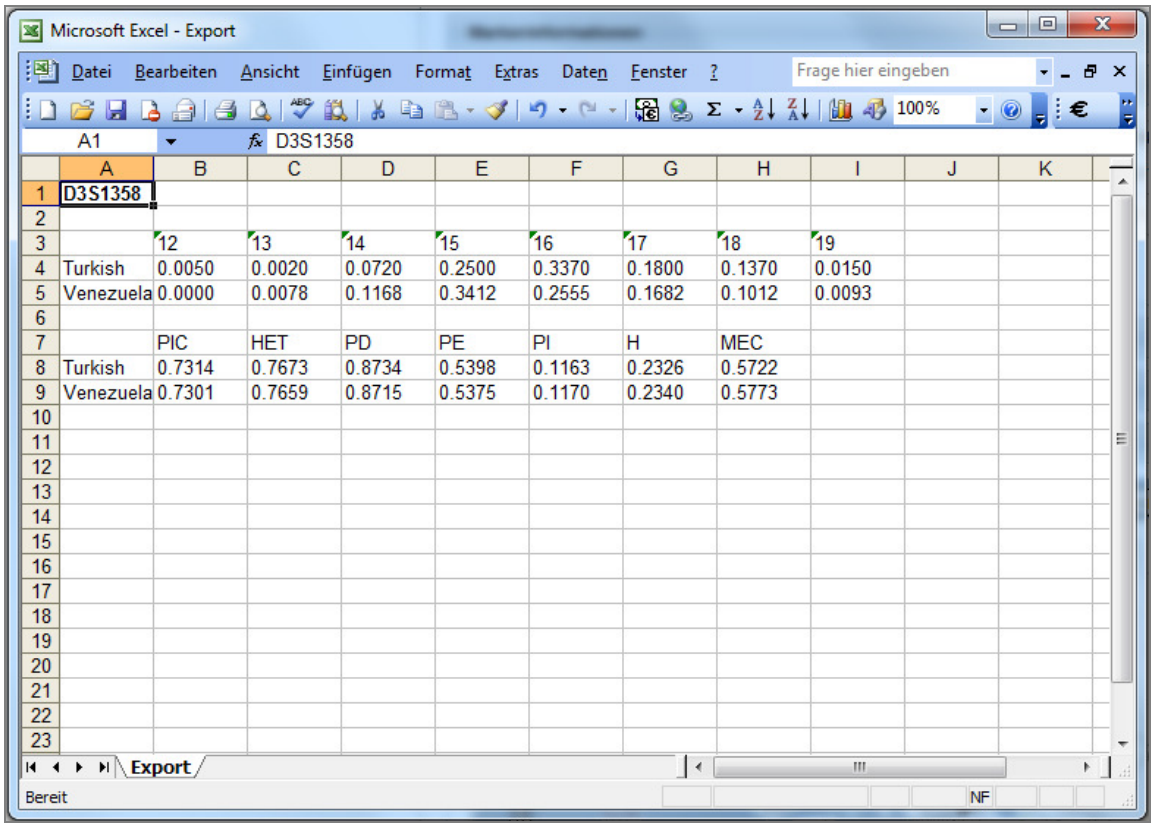


Abbildung 7-9: Exportierte Excel-Datei – erstellte Excel-Datei nach Export

8 Zusammenfassung

Im abschließenden Kapitel werden die bisher gewonnenen Ergebnisse zusammengefasst und eine Bewertung aus Sicht des Autors vorgenommen. Ein Ausblick zeigt Weiterentwicklungspotenziale der Anwendung auf.

8.1 Ergebnisse

Im Rahmen dieser Arbeit wurde ein Prototyp zur Verwaltung von genetischen Populationsdaten entwickelt. Der Prototyp bildet das Grundgerüst und umfasst dabei die folgenden Grundfunktionen einer typischen Datenbankanwendung: Daten importieren, Daten einsehen und Daten exportieren. Die Funktionalitäten der Datenbank können durch den modularen Aufbau beliebig ausgebaut und erweitert werden.

In der Anfangsphase wurde das biologische Problem in ein informationstechnisches Konzept transferiert. Während dieser Phase entstanden theoretische Geschäftskomponenten (Klassenobjekte) wie beispielsweise *Marker*, *Population* und Autor. Danach wurden die Zusammenhänge dieser Komponenten in eine Anwendungslogik eingebettet. Im Anschluss daran begannen die eigentlichen Programmierarbeiten. Als Ergebnis entstand ein Kreislauf, welcher die oben genannten Grundfunktionalitäten (Import, Einsehen und Export von Daten) vereint. Der Import von Daten erfolgt über Formulare. Der Nutzer kann alle datensatzrelevanten Parameter eingeben und diese an die Webanwendung übermitteln. Nach erfolgreicher Validierung werden die Daten in der Datenbank gespeichert. Um bereits publizierte Daten einzusehen, bieten sich dem Nutzer verschiedene Möglichkeiten an schnell und sicher an die gewünschten Datensätze zu gelangen. Als Hilfsmittel dienen Tabellen, eine Weltkarte sowie eine Schnellsuche. Die eigentlichen Datensätze werden dabei nicht „trocken“ präsentiert, sondern sind durch zusätzliche Informationen angereichert. Zusätzlich zu den Datensätzen werden biostatistische Berechnungsparameter angeboten. Ebenfalls findet der Nutzer Angaben über Referenzen zu den Datensätzen.

Die letzte Grundfunktion betrifft das Herunterladen von Daten. Durch einen Exportservice können Datensätze als *xls-Datei* auf den Homerechner für weitere Arbeiten heruntergeladen werden. Zusätzliche Funktionen, wie ein Anzeigebereich für Neuigkeiten und Statistiken zum System, runden die Funktionen des Prototyps ab.

Fazit

Wie oben erwähnt stellt der entwickelte Prototyp die Basisfunktionen für eine serienreife Umsetzung bereit. Um zukünftig konkurrenzfähig mit anderen Datenbanken im forensischen Bereich zu bleiben, sollten jedoch weitere Dienste und Funktionen, welche im Ausblick diskutiert werden, angeboten werden.

8.2 Ausblicke

Die Arbeiten an der Webanwendung sollten auf keinen Fall mit dem Abschluss der Arbeit eingestellt werden. Die folgenden Ausblicke von informationstechnischer sowie biologischer Seite sollen diese wissenschaftliche Arbeit abschließen:

Die **informationstechnische** Seite bietet viel Potenzial für Weiterentwicklung an. So sollte man es nicht bei den zwei Standardsprachen englisch und deutsch beruhen lassen. Die Integration zusätzlicher Sprachen, wie Spanisch und Französisch sollten sich ohne große Probleme mit Hilfe der *Properties-Dateien* durchführen lassen. Da die Mehrsprachigkeit derzeit nur auf Programmebene realisiert wurde, könnte ebenfalls ein Konzept für eine Mehrsprachigkeit auf Datenbankebene erarbeitet und umgesetzt werden.
Die Erfassung der Datensätze mit Hilfe der Formulare zum derzeitigen Standpunkt ist noch sehr zeitintensiv, von daher ist es ratsam einen zusätzlichen Mechanismus zum Import von Datensätzen anzubieten (z.B. *CSV-Import*). Auch der Exportservice sollte durch zusätzliche Formate wie PDF, *XML* oder OpenOffice erweitert werden. Mit steigender Anzahl an Datensätzen und somit einem steigenden Maß an Unübersichtlichkeit, ist ebenso die Implementierung eines erweiterten Suchmechanismus von Nöten. Weiterhin wäre es denkbar, die autosomale Datenbank mit einer *DNA*-Analyse-Software zu koppeln. Hierfür müsste ein Format, sowie klare Schnittstellen definiert werden, um ein Kommunikationsszenario zwischen Webanwendung und Softwareanwendung zu ermöglichen.

Auch von **biologischer** Seite sollte das System erweitert werden. Es sollten sogenannte *Haplotypen* im System integriert werden. Dabei sollte überlegt werden, ob ein neues Klassenmodell erarbeitet werden muss oder ob ein Lösungsansatz für das bestehende System möglich ist. Ebenso sollte die derzeitige Unterteilung der *Populationen*

nochmals überarbeitet werden. Zurzeit werden *Populationen* nur nach Kontinenten kategorisiert. Die Zuordnung der *Populationen* könnte jedoch noch feiner unterteilt werden. (z.B. Kontinent: Europa, *Population*: Deutsche, *Subpopulation*: Bundesländer (z.B. Sachsen)). Dafür müssten lediglich Teile des Klassenmodells abgeändert werden. Verschiedene Tools, wie die Berechnung von Identitätswahrscheinlichkeiten und Ähnlichkeitsberechnungen erweitern die Funktionspalette der Datenbank jederzeit und könnten in einen *Premium-Benutzer-System* mit kommerziellem Nutzen eingebettet werden.

Glossar

AJAX - Asynchronous JavaScript And XML bezeichnet ein Konzept der asynchronen Datenübertragung zwischen Web-Browser und Web-Server. Es findet oft Einsatz in *Rich Internet Applications* (*RIAs*).

AJAX-Request – Ein AJAX-Request ist eine spezielle Form des *HTTP-Requests* auf asynchroner Basis.

Allel - Ein Allel gibt die Anzahl der Muster-Wiederholungen an einem Marker an. Da beim Menschen jedes *Chromosom* doppelt vorhanden ist kann es zu jedem Marker zwei Allele geben (*Heterozygotie*). Im homozygoten Fall gleichen sich die vererbten Wiederholungen von Vater und Mutter und es existiert nur ein Allel an diesem Ort.

Allelfrequenzen – Allelfrequenz ist ein Begriff aus der Populationsgenetik. Sie stellen die relative Häufigkeit der Kopien eines *Allels* in einer *Population* dar. Die Allelfrequenzen beschreiben die genetische Vielfalt einer *Population*.

Arlequin-Datei – Eine Arlequin-Datei ist ein Dateiformat für die Software Arlequin, welche für die Analyse populationsgenetischer Daten genutzt wird.

Alleldiversität – Die Alleldiversität ist ein Maß für die genetische Vielfalt einer *Population*.

Annotations erlauben es Metadaten in den Quelltext einzubinden. Sie beginnen mit einem @-Zeichen und können beim Kompilieren eines Programms ausgewertet werden.

API – Application Programming Interface - Eine API ist eine Schnittstelle, die von einem Softwaresystem anderen Programmen zur Anbindung an das System zur Verfügung gestellt wird. Die Programmverbindung findet dabei auf Quelltextebene statt.

API-Key – Ein API-Key ist im Zusammenhang mit Google Maps ein generierter Schlüssel um einen Dienst von Google nutzen zu können.

Applikationsserver - Als Applikationsserver wird im Allgemeinen ein Server bezeichnet auf dem Geschäftsanwendungen ausgeführt sowie verschiedene Dienste bereit gestellt werden.

AOP - Aspektorientierte Programmierung – Die AOP stellt ein Paradigma dar nach dem in der Softwareentwicklung programmiert werden kann.

BigDecimal ist ein Datentyp zur Beschreibung eines Zahlenformates und ist wesentlich genauer wie der Datentyp *Double*.

BMP – Bean Managed Persistence ist ein Verfahren für die Persistierung von *Entity Beans*, bei diesem Verfahren ist der Entwickler für die Speicherung der *Entity Beans* verantwortlich.

build-Script - Ein build-Script ist eine Datei, welche alle nötigen Instruktionen zum Bau einer Anwendung benötigten Parameter und Einstellungen beinhaltet.

Calc ist ein Tabellenkalkulationsprogramm von der Firma Oracle und gehört zur Klasse der Office-Software Open Office.

CheckBox-Komponente – Eine CheckBox-Komponente ist ein Standardbedienelement grafischer Benutzeroberflächen. Mit ihr werden oft ja/nein-Fragen beantwortet.

Chromatin ist das Material aus dem die *Chromosomen* bestehen. Es handelt sich um einen Komplex aus *DNA* und *Proteinen*.

Chromosomen sind Strukturen im Zellkern von *Eukaryoten*, welche die *Gene* und damit die Erbinformation enthalten.

CMP – Container Managed Persistence ist ein Verfahren für die Persistierung von *Entity Beans,* bei diesem Verfahren ist die Laufzeitumgebung für die Speicherung der *Entity Beans* verantwortlich.

Composition-Tag – Das Composition-Tag gehört zur *Facelets-Tag Library* und wird für die Verwendung von Templates benutzt.

Container oder auch *EJB*-Container ist eine Software die auf einem *Applikationsserver* läuft und *Enterprise JavaBeans* verwaltet.

Conversiational State wird der an einen Client gebundene Status einer *Session Bean* genannt.

ConvertNumber-Tag – Das ConvertNumber-Tag gehört zur *JSF-Tag Library* und dient zur Konvertierung von Zahlen.

CRUD-Anwendung – Eine CRUD-Anwendung ist eine Anwendung, welche aus existierenden Tabellen einer Datenbank erstellt wurde und die Grundfunktionalitäten Daten anlegen, Daten abfragen, Daten aktualisieren und Daten löschen beinhaltet.

CSS - Cascading Style Sheets ist eine deklarative Stylesheet-Sprache für strukturierte Dokumente. Sie wird vor allem zusammen mit *HTML* und *XML* eingesetzt.

CSV-Import ist ein spezielles Format zum Import von Daten, dabei werden die Datensätze per Kommata getrennt.

Datascroller – Datascroller sind Komponenten zur Unterstützung von *Pagination*.

DataTable-Tag – Das DataTable-Tag gehört zur *RichFaces-Tag Library* und wird genutzt um Daten in Form von Tabellen auszugeben.

Debugging ist eine Möglichkeit in der Softwareentwicklung Fehler aufzufinden und zu diagnostizieren.

Decorate-Tag - Das Decorate-Tag hat die gleiche Funktion wie das *Composition-Tag*. Der einzige Unterschied ist, dass die Texte außerhalb des Tags, für die Ausgabe berücksichtigt werden.

Define-Tag – Das Define-Tag gehört zur *Facelets-Tag Library* und ist das Gegenstück zum *Insert-Tag*. Das Define-Tag kann nur innerhalb eines *Composition-* oder *Decorate-Tag* verwendet werden.

DI - Dependency Injection ist ein Entwurfsmuster und dient in einem objektorientierten System dazu, die Abhängigkeiten von Komponenten oder Objekten zu minimieren.

Deployment – Als Deployment bezeichnet man den Prozess, bei dem die Webanwendung in eine Web-Server-Umgebung installiert wird.

Desoxyribose bezeichnet ein Zuckermolekül, dass aus fünf Kohlenstoff-Atomen besteht und somit das Zuckergerüst der *DNA* bildet.

Div-Tag – Das Div-Tag gehört zur *HTML-Tag Library* und ist ein Behälter für weitere *HTML*-Elemente.

DNA – Desoxidribonukleinsäure – DNA ist die Bezeichnung für den chemischen Aufbau der Erbinfomationen. Sie besteht aus den vier Basen Adenin, Thymin, Cytosin und Guanin.

Drag&Drop ist eine Methode zur Bedienung von grafischen Benutzeroberflächen. Dabei können Elemente (z.B. Symbole, Bilder) gezogen und an einem bestimmten Ziel wieder abgesetzt werden.

Drools siehe *Rules*

DRY-Konzept - Don´t repeat yourself-Konzept bezeichnet ein Prinzip in der Softwareentwicklung, das besagt, Redundanz zu vermeiden oder zumindest zu reduzieren.

Double ist ein Datentyp für die Bezeichnung eines Zahlenformates.

Drei-Tier-Architektur ist eine Architektur die softwareseitig drei Schichten besitzt. Die Drei-Tier-Architektur besteht meistens aus Präsentations-, Logik- und Datenhaltungsschicht.

DropDown-Menü ist ein Navigationsmenü, welches dem Benutzer durch eine Aktion (z.B. Mausklick) auf ein Hauptelement die einzelnen Unterelemente anzeigt.

EAR-Datei – Eine EAR-Datei bezeichnet ein Format, welches eine *Java Enterprise Edition* Anwendung beinhaltet.

Eclipse ist ein Werkzeug zur Erstellung von Software und wird in den Bereich *IDE* eingegliedert.

EJB-Container – Ein EJB-Container ist eine spezielle Form eines *Conatiners* für *Enterprise JavaBeans*.

EJB – Enterprise JavaBeans sind Komponenten aus der Spezifikation *Java Enterprise Edition* und stellen eine spezielle Form von *JavaBeans* dar.

Entity Beans gehören zum *EJB*-Konzept und modellieren persistente Anwendungsobjekte.

Entity Manager verwaltet die Zuordnung (mapping) zwischen relationalen Datenbanktabellen und *Entity Bean*-Objekten.

Eukaryoten - Als Eukaryoten werden alle Lebewesen mit Zellkern und Kernmememembran zusammengefasst. Eukaryoten besitzen mehrere *Chromosomen*.

Facelets ist eine alternative View-Technologie für *Java ServerFaces*. Facelets kann *Java ServerPages* als Definitionssprache für die View ersetzen.

FacesMessages ist eine Komponente aus der *Seam-Tag Library*, welche es ermöglicht Meldungen auf einer Website auszugeben.

Gelelektrophorese ist eine analytische Methode der Molekularbiologie um verschiedene Arten von Molekülen der Größe nach zu trennen.

Gen - Ein Gen ist ein Abschnitt auf der *DNA*. Manche Gene bestimmen allein über eine Eigenschaft wie z.B. die Augenfarbe.

GenoProof®2 ist eine DNA-Analyse-Software für Vaterschaftsuntersuchungen, Verwandtschaftsanalysen und zur Auswertung von Populationsstudien.

Genotyp – Ein Genotyp bezeichnet alle in der *DNA* codierten genetischen Informationen.

Getter-/Setter-Methoden sind spezielle Methoden einer *JavaBean* um auf deren Attribute zuzugreifen (Get) oder ändern (Set) zu können.

Haplotyp ist eine Kombination von *Genotypen* mit zwei oder mehreren *Markern* desselben elterlichen *Chromosoms*.

HET – Heterozygotie ist ein Begriff aus dem Bereich der Genetik und bedeutet Mischerbigkeit.

Hibernate ist ein Open-Source-Persistenz-Framework für Java. Es unterstützt die *Object-Relational Mapping*-Funktionalität.

h – Homozygotie ist ein Begriff aus dem Bereich der Genetik und bedeutet Reinerbigkeit.

HTML – Hyper Text Markup Language ist das Format, in dem die Text- und Hypertext-Informationen im World Wide Web gespeichert und übertragen werden.

HTTP – Hyper Text Transfer Protocol ist ein Protokoll zur Übertragung von Daten über ein Netzwerk.

HTTP-Request ist die Anfrage eines Clients, nach Verbindungsaufbau, an den Server.

HTTP-Response bezeichnet die Antwort eines Servers auf die Anfrage (*HTTP-Request*) eines Clients.

Idiogramme sind ein Hilfsmittel für die grafische Darstellung von *Chromosomen*.

IDE - Integrated Development Environment ist ein Anwendungsprogramm zur Erstellung von Software.

Include-Tag – Das Include-Tag gehört zur *Facelets-Tag Library* und wird benutzt um eine Webseite einzubetten.

InputNumberSpinner-Komponente – Die InputNumberSpinner**-**Komponente gehört zur *RichFaces-Tag Library* und ermöglicht es eine Zahl einzustellen.

Insert-Tag gehört zur *Facelets-Tag Library* und wird verwendet um Templates zu definieren. Durch dieses Tag werden bestimmte Bereiche definiert, die von anderen Seiten überschrieben werden können.

Interface - Ein Interface in Java ist die Möglichkeit zu einer oder mehreren Klassen bzw. von einem oder mehreren anderen Interfaces eine Schnittstelle als neue, komplexe Signatur zu definieren und anzuwenden.

IoC - Inversion of Control bezeichnet ein Umsetzungsparadigma in der objektorientierten Programmierung. Ein Entwickler beschreibt nur die Abhängigkeiten zu anderen Anwendungskomponenten.

ISFG - International Society for Forensic Genetics ist eine internationale Vereinigung aus den Gebieten der Genetik und Forensik.

JAR-Datei – Eine JAR-Datei ist eine gepackte Datei, welche verschiedene Java-Dateien bündelt.

JavaBeans sind Java-Klassen, die einem Komponentenmodell entsprechen, um automatisierten Zugriff auf deren Eigenschaften und Operationen zuzulassen.

Java EE – Java Enterprise Edition ist eine offene Spezifikation einer Softwarearchitektur für die transaktionsbasierte Ausführung von in Java programmierten Anwendungen.

JBoss AS - JBoss Application Server ist ein *Applikationsserver* von der Firma Red Hat.

JBoss Tools sind Werkzeuge für *Eclipse* zur Administration von JBoss-Technologien (z.B. *JBoss Applikation Server*, JBoss *RichFaces*).

Rules ist ein Security-Framework von der Firma JBoss für das Organisieren von Regeln, Workflows und Events.

jBPM – Java Business Process Management ist eine Plattform zur Ausführung mittels ausführbarer Geschäftsprozesssprachen definierter Arbeitsabläufen aus Java.

JExcel unterstützt den Entwickler bei der Erstellung von Dateien im *xls-Format* für Microsoft Excel.

JPDL - jBPM Process Definition Language ist eine proprietäre Process Execution Language (Prozessausführungssprache), welche ausschließlich von der auf Java basierenden Process Engine *jBPM* interpretiert wird.

JRE – Java Runtime Environment – Die JRE enthält alle Komponenten, die für die Ausführung von Java-Programmen benötigt werden. Dazu gehören die Java Virtual Machine und Java-Klassenbibliotheken.

JSF - Java ServerFaces ist ein Framework und bietet die Möglichkeit auf einfache Art und Weise Komponenten für Benutzerschnittstellen in Internetseiten einzubinden.

JSP – Java ServerPages ist eine Spezifikation zur Erstellung von dynamischen Internetseiten. Die JSP-Dateien bestehen aus statischen und dynamischen Textelementen.

Java SE - Java Standard Edition ist eine Sprachspezifikation von Java und umfasst eine Sammlung verschiedener Java-APIs.

JPA – Java Persistence API ist eine Schnittstelle für Java-Anwendungen, die die Zuordnung und die Übertragung von Datenbankeinträgen vereinfacht.

JSF EL – JSF Expression Language ist eine Beschreibungssprache für *JSF* um auf Geschäftsdaten zugreifen zu können.

Komponentenbibliothek siehe *Tag Library*

Locus siehe *Marker*

Marker - Als Marker bezeichnet man in der Molekularbiologie eindeutig identifizierbare, kurze *DNA*-Abschnitte, deren Ort im Erbgut bekannt ist.

MEC - Mean Exclusion Chance ist ein biostatistischer Berechnungsparameter welcher Auskunft darüber gibt ob ein möglicher Elternteil von der Elternschaft ausgeschlossen werden kann.

MDB - Message Driven Bean sind spezielle *JavaBeans*, welche *EJB*-Systeme für den asynchronen Nachrichtenaustausch zugänglich machen.

Message-Schlüssel – Ein Message-Schlüssel wird in *Properties-Dateien* verwendet um auf einen beliebigen Wert zuzugreifen.

MVC-Modell – Model-View-Controller-Modell ist ein Architekturmuster zur Strukturierung von Softwareprojekten in die drei Einheiten: Model, View und Controller.

NetBeans ist ein Werkzeug zur Erstellung von Software und wird in den Bereich *IDE* eingegliedert.

ORM - Object-Relational Mapping ist eine Technik in der Softwareentwicklung mit der Objekte in einer relationalen Datenbank abgelegt werden können.

ORM-Mapper - Object-Relational Mapper – Der ORM-Mapper ist eine Software, welche *Object-Relational Mapping* unterstützt.

Packaging nennt man das Verpacken einer Webanwendung in eine *WAR*- oder *EAR-Datei*.

Pagination - Sobald man sehr langen Seiteninhalt hat oder die Anzahl der Artikel soweit angestiegen ist, dass der User die Internetseite endlos nach unten scrollen muss, sollte man den Inhalt oder die Artikel auf mehrere Seiten aufteilen. Diese Funktionalität wird Pagination genannt und erhöht die Benutzerfreundlichkeit.

Panel sind in der Softwareentwicklung einfache, grafische Gestaltungselemente die mit einem Layoutmanager positioniert werden.

PCR - Polymerase-Kettenraktion ist eine Methode um die Erbsubstanz (*DNA*) zu vervielfältigen. Dazu wird ein Enzym verwendet, die *DNA*-Polymerase. Der Begriff Kettenreaktion beschreibt in diesem Zusammenhang die Tatsache, dass die Produkte vorheriger Zyklen als Ausgangsstoffe für den nächsten Zyklus dienen und somit eine exponentielle Vervielfältigung ermöglichen.

PD – Power of Discrimination ist ein biostatistischer Berechnungsparameter und trifft eine Aussage darüber wie gut sich zwei Menschen voneinander unterscheiden lassen.

pgAdminIII ist ein grafisches Adminstrationstool für *PostgreSQL*.

PHP - Hypertext Preprocessor ist eine serverseitige Scriptsprache zur Erstellung von dynamischen Webanwendungen.

PI – Paternity Index – Der PI ist ein biostatistischer Berechnungsparameter der oft bei der Auswertung von Vaterschaftstests auftaucht.

Poolingverwaltung - Um Zugriffe auf Instanzen von *Enterprise JavaBeans*-Komponenten (*EJB*-Komponenten) zu beschleunigen, nutzen *EJB-Conatiner* eine Poolingverwaltung. Clients erhalten so einen schnelleren Zugriff auf *EJBs*.

Population - Eine Population enthält möglichst viele *Genotypen* einer bestimmten Bevölkerungsgruppe (deutsch, türkisch etc.). Sie ermöglicht damit Aussagen zur *Allelefrequenz*, d.h. der Auftrittshäufigkeit von *Allelen* innerhalb der Bevölkerungsgruppe.

PostgreSQL ist ein freies, objektrelationales Datenbankmanagementsystem.

Premium-Benutzer-System - Ein Premium-Benutzer-System unterscheidet sich von einem Gratis-Benutzer-System dahingehend, dass der Benutzer für die Nutzung einiger Funktionalitäten Geld bezahlen muss.

Primer sind kurze *DNA*-Fragmente, die im *Testkit* enthalten sind und während der *PCR* an bestimmten Markern eines *DNA*-Stranges andocken. Dadurch sind die zu isolierenden und zu vervielfältigenden *STR*-Regionen bestimmt.

Properties-Datei ist eine Textdatei, die in der Programmiersprache Java als einfacher Konfigurationsmechanismus verwendet wird. Eine Property ist in diesem Zusammenhang ein Text, der unter einem bestimmten Namen abgelegt ist.

Proteine sind aus Aminosäuren aufgebaute Makromoleküle, welche in ihrer Grundsubstanz aus Kohlenstoff, Wasserstoff, Sauerstoff, Stickstoff und Schwefel bestehen. Proteine gehören zu den Grundbausteinen aller Zellen.

Putativelternteil wird derjenige bezeichnet, der als möglicher Elternteil eines Kindes in Frage kommt. Hatte eine Frau im Zeitraum der Befruchtung Geschlechtsverkehr mit zwei Männern, werden beide Männer als Putativvater bezeichnet.

Repeat-Komponente – Die Repeat-Komponente gehört zur *RichFaces*-Tag Library und kann z.B. ein Array iterativ durchlaufen.

RIA - Rich Internet Application beschreibt eine Anwendung, welche die Internet-Technik benutzt und eine mächtige Benutzeroberfläche bietet.

RichFaces von der Firma JBoss ist eine *Komponentenbibliothek* für *Java ServerFaces*.

Seam ist ein Framework zur Erstellung von Webanwendungen mit Hilfe von *Java EE*-Technologien von der Firma JBoss.

seam-gen ist ein von *Seam* mit ausgelieferter Generator zur Generierung von Projekten mit kompletter *Eclipse*- und *NetBeans-IDE*-Unterstützung.

SelectOneMenu-Komponente ist eine *UI-Komponente* aus der *JSF-Tag Library* und bietet dem Benutzer mehrere Einträge zur Auswahl an.

Servlet – Als Servlet wird eine Java Klasse bezeichnet, deren Instanzen innerhalb eines Java-Web-Servers Anfragen von Clients entgegen nehmen und beantworten.

Servlet-Engine - Die *Servlet*-Engine ermöglicht dem Web-Server das Ausführen von *Servlets*.

Session – Als Session bezeichnet man eine stehende Verbindung zwischen einem Client und einem Server.

Session Beans gehören zum *EJB*-Konzept und realisieren die Logik des Geschäftsprozesses.

Sequenzierautomat - Nach der *PCR* trennt der Sequenzierautomat die *DNA*-Fragmente der Länge nach. Dies geschieht mittels *Gelelektrophorese*.

Spring ist ein Framework zur Erstellung von Java Anwendungen von der Firma Spring Source.

Spring MVC ist ein Teil-Framework von *Spring* zur Erstellung von Webanwendungen.

Spring Security ist ein Teil-Framework von *Spring* zur Absicherung von Java-Anwendungen.

Spring Web Flow ist ein Teil-Framework von *Spring* für die Implementierung von Abläufen (Navigation) in einer Website.

STR – Short Tandem Repeats sind Wiederholungen von bestimmten Basenmustern. Die Anzahl der Wiederholungen variiert dabei stark zwischen verschiedenen Individuen.

Subclipse ist ein Tool, welches die Entwicklungsumgebung *Eclipse* mit der Versionsverwaltung *Subversion* koppelt.

Subpopulation – Eine Subpopulation ist die Unterteilung einer *Population* nach geografischer Zugehörigkeit und orientiert sich u.a. an den jeweiligen Sprachgrenzen.

Subversion ist ein Tool zur Versionsverwaltung von Softwareprojekten.

SuggestionBox-Komponente – Die SuggestionBox-Komponente gehört zur *RichFaces-Tag Library* und ermöglicht es auf Eingaben eines Benutzers in ein Textfeld zu reagieren.

Syntax Highlithing bezeichnet die Möglichkeit einer Software bestimmte Wörter und Zeichenkombinationen in einem Text abhängig von ihrer Bedeutung, in unterschiedlichen Farben, Schriftarten und –stile darzustellen.

Tag Libraries sind definierte Tags (z.B. für *HTML*), die durch eine Engine (z.B. *Java ServerFaces*) verarbeitet werden. Dies dient der Trennung von Code und Design.

TCP-/IP ist ein Protokoll, welches während einer Verbindung den Datenaustausch zwischen den Computern regelt.

Testkit - Ein Testkit enthält die sogenannten *Primer*, welche die zu untersuchenden *DNA*-Orte (*Marker*) lokalisieren und isolieren. Testkits enthalten auch die Fluoreszenzfarbstoffe, mit denen die Fragmente sowie der Größenstandard während der *PCR* markiert werden.

TestNG ist ein Framework zum Testen von Anwendungen.

Threadverwaltung bezeichnet die Verwaltung von mehreren leichtgewichtigen Prozessen.

UI-Komponente - UserInterface-Komponenten - Als UI-Komponenten werden Objekte bezeichnet, die die Interaktion eines Benutzers mit einer Anwendung ermöglichen.

URL – Uniform Resource Locator identifiziert und lokalisiert eine Ressource über das verwendete Netzwerkprotokoll (z.B. *HTTP*).

UserInterface-Eventhandler-Methoden sind Methoden von *UI-Komponenten*, die ausgelöst werden, wenn ein bestimmtes Event ausgelöst wird (z.B. Drücken eines Buttons)**.**

Use Case - Ein Use Case beschreibt einen möglichen Anwendungsfall in einer Anwendung den ein Benutzer durchführt um ein bestimmtes Ziel zu erreichen.

Visual Editor ist im Zusammenhang mit *JSF* ein Editor zum visuellen betrachten von *JSF*-Webseiten.

WAR-Datei – Eine WAR-Datei ist ein Dateiformat, welches alle benötigten Dateien einer Webanwendung bündelt.

XHTML – Extensible Hyper Text Markup Language ist eine *XML*-konforme Weiterentwicklung von *HTML*.

xls-Format – sind Dateien für das Softwareprodukt von Microsoft Excel.

XML – Extensible Markup Language ist eine Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten in Form von Textdaten.

Literaturverzeichnis

[1] Bundesminsterium für Bildung und Forschung: *Chromosomen*. http://www.ngfn.de/de/glossar.html?chrLetter=C&strTermsub=Chromosmen, Abruf 26.05.2010. Internet

[2] Unbekannt: *Bau und Funktion der DNA*. http://www.zum.de/Faecher/Materialien/beck/bs11-70.htm, Abruf 24.05.2010. Internet

[3] Unbekannt: *Aufbau des Chromosoms und der DNA*. http://tgg-leer.de/projekte/genetik/dna2/dna2.html, Abruf 02.06.2010. Internet

[4] Wikipedia: *Eukaryoten*. http://de.wikipedia.org/wiki/Eukaryoten, Abruf 02.06.2010. Internet

[5] Wikipedia: *Chromosomen*. http://de.wikipedia.org/wiki/Chromosom, Abruf 01.06.2010. Internet

[6] Götz, Frank: *Anwendung der Bio-Informatik in der Forensik*, Qualitype AG, Dresden, 2008

[7] Nemak, Sebastian: *Projektdokumentation zur betrieblichen Projektarbeit, Webanwendung zur Verwaltung von X-Chromosomalen-Markern für den forensischen Zweck*, Qualitype AG, Dresden, 2009

[8] Deutsches Hygiene Museum Dresden: *Polymerase-Kettenreaktion (PCR) &Gelelektrophorese*.http://www.dhmd.de/neu/fileadmin/template/dhmd/images/uploads/glaesernes_labor/PDF_Dokumente/PCR_Elektrophorese051109.pdf, Abruf 14.06.2010. Internet

[9] Götz, Frank: *GenoProof®2*. http://qualitype.de/genoproof/, Abruf 10.06.2010. Internet

[10] Scheil, Hans-Georg; Huckenbeck, Wolfgang: *DNA: Ein Crash-Kurs (3) für Einsteiger.* http://www.uni-duesseldorf.de/WWW/MedFak/Serology/seronews-Dateien/teil_3.pdf, Abruf 22.06.2010. Internet

[11] Wikipedia: *Homozygotie*. http://de.wikipedia.org/wiki/Homozygotie, Abruf 20.06.2010. Internet

[12] Wikipedia: *Heterozygotie*. http://de.wikipedia.org/wiki/Heterozygotie, Abruf 20.06.2010. Internet

[13] Unbekannt: *Village Dog Diversity Project*. http://villagedogs.canmap.org/tutorials/microLesson.aspx, Abruf 21.06.2010. Internet

[14] Götz, Frank: *Theoriehandbuch GenoProof®2*, 1. Aufl., Qualitype AG, Dresden, 2010

[15] Buttler, John M.: *Forensic DANN Typing, Second Edition: Biology, Technology, and Genetics of STR Markers*, 2. Aufl., Elsevier Academic Press, Elsevier (USA), 2005

[16] John, Cordula; Berger, Anna-Antonia: *Funktionsspezifikation Autosomale Datenbank*, Qualitype AG, Dresden, 2009

[17] Unbekannt: *An Integrated Software for Population Genetics Data Analysis*. http://cmpg.unibe.ch/software/arlequin3/, Abruf 25.06.2010. Internet

[18] Allele Frequency Database: *„Alfred" Highlights*. http://alfred.med.yale.edu, Abruf 25.05.2010. Internet

[19] Unbekannt: *Rich Internet Applications*. http://www.xing.com/net/ria/, Abruf 12.10.2010. Internet

[20] Gündel, Andreas: *Access Protection in a Distributed Smalltalk-System*. http://www.heeg.de/~georg/guendel.htm, Abruf 28.06.2010. Internet

[21] Zimmer, Frank: *Web-Programmierung II, Java-Basistechnologien für webbasierte Anwendungen* https://www.htwm.de/~mmlab/intranet/doc_download.php?file=051215170736.pdf&title=WebProg_Lehrunterlagen_04_TeilModelle.pdf, Abruf 24.06.2010. Internet

[22] Haas, Roland; Schreiner Ulrich: *Java-Technologien für Unternehmensanwendungen Konzepte, Methoden und Anwendungen der Java 2 Enterprise Platform*, Carl Hanser Verlag, München, 2002

[23] Unbekannt: *Model-View-Controller in Java*. http://arturleinweber.de/schule/informatik/model_view_controller_java.php, Abruf 13.06.2010. Internet

[24] Wikipedia: *Webanwendung*. http://de.wikipedia.org/wiki/Webanwendung, Abruf 20.06.2010. Internet

[25] Oracle: *Java EE at a Glace*. http://www.oracle.com/technetwork/java/javaee/overview/index.html, Abruf 15.06.2010. Internet

[26] Oracle: *Java EE 6 Technologies*. http://www.oracle.com/technetwork/java/javaee/tech/index.html, Abruf 15.06.2010. Internet

[27] Brendtner, Thomas: *Konzepte und Technologien für verteilte Systeme*. http://swt.cs.tu-berlin.de/lehre/sepr/ss08/referate/VerteilteSysteme_Folien-.pdf, Abruf 02.06.2010. Internet

[28] Oracle: *Enterprise Java Beans Technlogie*. http://www.oracle.com/technetwork/java/index-jsp-140203.html, Abruf 05.06.2010. Internet

[29] Wikipedia: *Ajax (Programmierung).* http://de.wikipedia.org/wiki/Ajax_(Programmierung), Abruf 26.06.2010. Internet

[30] Garett, James: *A New Approach To Web Applications*. http://www.adaptivepath.com/ideas/essays/archives/000385.php, Abruf 27.06.2010. Internet

[31] Heute, Thomas; Yuan, Michael: *JBoss Seam, Enterprise-Webanwendungen mit Java EE-einfacher und leistungsstärker*, 1. Aufl., Redline GmbH, Österreich, 2008

[32] JBoss: *JBoss Seam*. https://www.jboss.com/products/seam/, Abruf 03.07.2010. Internet

[33] Müller, Bernd: *JBoss Seam, Die Web-Beans-Implementierung*, Carl Hanser Verlag, München, 2008

[34] Ratzlow, Frank; Groth, Jan: *Klebstoff für Java EE*. In: Javamagazin (2009), Januar, 01/2009, Seite 40-44

[35] Spring Source: *Spring Source Community*. http://www.springsource.org/about, Abruf 01.07.2010. Internet

[36] Oates, Richard; Langer, Thomas; Wille, Stefan; Lueckow, Torsten; Bachlmayr, Gerald: *Spring & Hibernate, Eine praxisbezogene Einführung*, Carl Hanser Verlag, München, 2007

[37] Wikipedia: *Don´t repeat yourself.* http://de.wikipedia.org/wiki/Don´t_repeat_yourself, Abruf 28.06.2010. Internet

[38] Mjartan, Carsten: *Einführung in Spring Web Flow*. http://www.codecentric.de/__resources/pdf/artikel/einfuehrung-in-spring-web-flow.pdf, Abruf 15.07.2010. Internet

[39] Wiesener, Mike: *Enterprise Security mit Spring Security*. http://www.slideshare.net/mike.wiesner/enterprise-security-mit-spring-security-presentation, Abruf 18.17.2010. Internet

[40] Mann, Kito: *Java ServerFaces in Action*, Manning Publications Co., USA, 2005

[41] Katz, Max: *Practical RichFaces*, Springer-Verlag (Apress), USA, 2008

[42] JBoss: *JBoss Commuinty*. http://www.jboss.org/richfaces, Abruf 13.07.2010. Internet

[43] Beranek, Christian; Kneissl Isabella: *AJAX mischt mit, Java ServerFaces und AJAX*. http://christianberanek.de/wp-content/uploads/2008/06/jsfmitajax.pdf, Abruf 08.07.2010. Internet

[44] Oracle: *Unified Expression Language*. http://java.sun.com/products/jsp/reference/techart/unifiedEL.html, Abruf 22.07.2010. Internet

[45] Gühring, Philipp: *Zahlendarstellung in Programmen und nummerische Probleme*. http://www3.futureware.at/artikel/zahlen.htm, Abruf 14.07.2010. Internet

[46] Statista: *Verbreitung von Office-Software bei Internetnutzern in Deutschland im Januar 2010*. http://de.statista.com/statistik/daten/studie/77226/umfrage/internetnutzer---verbreitung-von-office-software-in-deutschland/, Abruf 11.10.2010. Internet

[47] Rätzmann, Manfred: *Session D-Tier, Three-Tier-Development*. http://www.dfpug.de/konf\konf_1998\09_tier\d_tier/d_tier.htm, Abruf 27.07.2010. Internet

[48] Unbekannt: *Unified Modeling Language*. http://www.omg.org/spec/UML/, Abruf 24.07.2010. Internet

[49] Assisi, Ramin: *Einführung und Referenz, Eclipse, Java-Entwicklung mit der Open-Source-Platform*, Carl Hanser Verlag, München, 2004

[50] Eclipse: *Eclipse Foundation*. http://www.eclipse.org, Abruf 02.06.2010. Internet

[51] CollabNet: *Das Subclipse Plugin für die Eclipse IDE*. http://subclipse.tugris.org/, Abruf 02.06.2010. Internet

[52] Unbekannt: *Subversion in Eclipse mit Subclipse nutzen*. http://www.admin-wissen.de/eigene-tutorials/programmierung/eclipse-workshop/workshop-seite-15/, Abruf 08.07.2010. Internet

[53] CollabNet: *Open Source Software Engineering Tools*. http://subversion.tigris.org, Abruf 04.06.2010. Internet

[54] JBoss: *JBoss Tools Plugin für die Eclipse IDE*. http://www.jboss.org/tools/, Abruf 07.06.2010. Internet

[55] Red Hat: *JBoss Enterprise Middleware*. http://www.redhat.com/jboss/, Abruf 05.06.2010. Internet

[56] Kühle, Markus: *Tutorial: Mit JBoss Seam und JEE5 unter Eclipse starten*. http://javathreads.de/2008/09/tutorial-mit-jboss-seam-und-jee5-unter-eclipse-starten, Abruf 13.06.2010. Internet

[57] PostgreSQL: *PostgreSQL*. http://www.postgresql.org, Abruf 10.06.2010. Internet

[58] Wikipedia: *SQL*. http://de.wikipedia.org/wiki/SQL, Abruf 23.07.2010. Internet

[59] pgAdmin: *PGAdmin*. http://www.pgadmin.org, Abruf 19.06.2010. Internet

[60] SelfHTML: *Allgemeine Elemente für Textbereiche*. http://de.selfhtml.org/html/text/bereiche.htm, Abruf 17.06.2010. Internet

[61] Google: Google Maps. http://maps.google.com, Abruf 02.10.2010. Internet

[62] Google: *Google Inc.*. http://www.google.de/intl/de/about.html, Abruf 13.10.2010

[63] Unbekannt: *Java Excel API – A Java API to read, write, and modify Excel spreadsheets*. http://jexcelapi.sourceforge.net/, Abruf 03.10.2010. Internet

[64] Zeller, Alexander: *Excel-Tools unter Java im Vergleich, JExcelAPI und Jakarta POI – David gegen Goliath?*. http://www.ordix.de/ORDIXNews/1_2008/Java_J2EE_JEE/jexcelapi_jakarta.html, Abruf 02.10.2010. Internet

[65] Unbekannt: *TestNG*. http://testng.org/doc/index.html, Abruf 06.10.2010. Internet

[66] Kühle, Markus: *Einführung in Unit Tests mit TestNG unter Eclipse*. http://javathreads.de/2009/02/einfuehrung-in-unit-tests-mit-testng-unter-eclipse/, Abruf 06.10.2010. Internet

[67] Unbekannt: *Browser Statistics*. http://www.w3schools.com/browsers/browsers_stats.asp, Abruf 12.10.2010. Internet

[68] Frank, Klaus: *Handbuch zum Testen von Web-Applikationen: Testverfahren, Werkzeuge, Praxistipps*, Springer Verlag, Berlin, 2007

[69] Carracedo, Angel: *Editorial Publication of population data for forensic purposes*. http://www.elsevier.com/framework_products/promis_misc/FSI http _Editorial.pdf, Abruf 28.04.2010. Internet

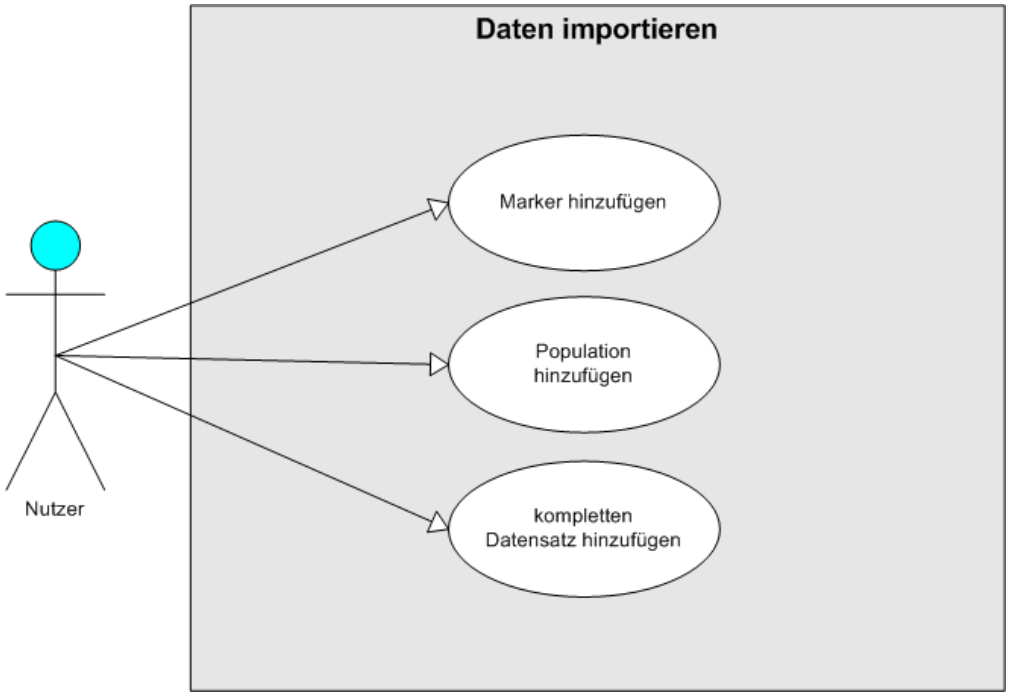
Anlagen, Teil 1

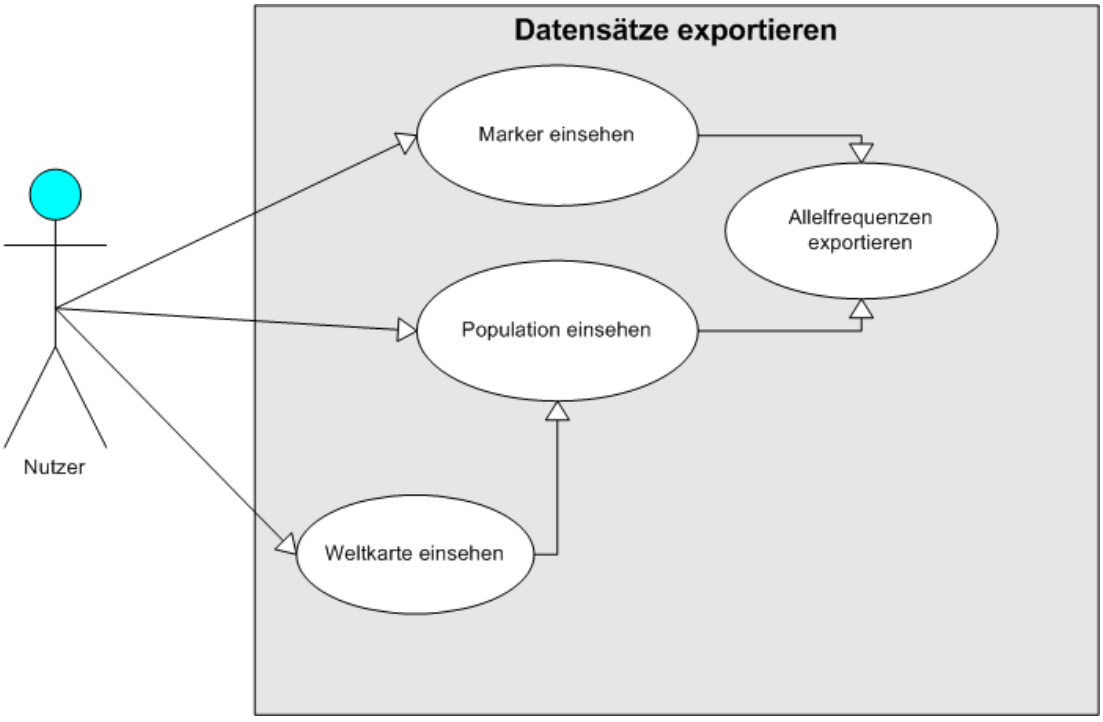
Die folgende Tabelle wurde während der Analyse der autosomalen Datenbanken angefertigt.

	The Allele Frequency Database	Autosomal STR DNA Database	"Huckenbeck" Database
URL	# http://alfred.med.yale.edu/	# http://www.strdna-db.org/	# http://www.uni-duesseldorf.de/WWW/MedFak/Serology/database.html
Aktualität	# viele Datensätze # Seite wird gepflegt und kontinuierlich aktualisiert	# wenige Datensätze # Arbeiten an Seite eingestellt	# wenige Datensätze # letzter Datensatz Oktober 2006
Navigation	# Drop Down Menü # Seitenstruktur sehr verwirrend # sehr komplexe Navigation	# Menü im Kopfbereich (Links) # gut navigierbar, keine Komplexität	# kein Menü # gut navigierbar, keine Komplexität
Usability	# komplizierte Bedinung # Hilfe vorhanden # Suchmasken vorhanden	# keine Hilfe # keine Suchmasken # einfache Bedinung	# keine Hilfe # keine Suchmasken # einfache Bedinung
Weltkarte	# Weltkarte vorhanden # viele Tabellen und Diagramme	# Weltkarte vorhanden # Tabellen	# keine Weltkarte # keine visuellen Features
Mehrsprachigkeit	# keine Mehrsprachigkeit # Standardsprache Englisch	# keine Mehrsprachigkeit # Standardsprache Englisch	# keine Mehrsprachigkeit # Standardsprache Englisch
Marker-informationen	# viele Zusatzinformationen # Angabe von Referenzen # Synonyme # Lokalisierungsangaben	# keine Zusatzinformationen # keine Referenzen	# keine Zusatzinformationen # Angabe von Referenzen
Populations-informationen	# viele Zusatzinformationen zu den Populationen # Angabe von Referenzen # geographische Lokalisierung # Link zu pubMed	# keine Zusatzinformationen # keine Referenzen	# keine Zusatzinformationen # Angabe von Referenzen
Datensätze	# Datensätze sehr versteckt # viele Zusatzinformationen zu den Datensätzen	# Datensätze können nur exportiert werden (Excel) # keine Zusatzinformationen	# Datensätze schnell erreichbar # keine Zusatzinformationen
Qualitätskontrolle	# keine strenge Qualitätskontrolle	# keine strenge Qualitätskontrolle	# keine strenge Qualitätskontrolle
biostatistische Berechnungen	# keine vorhanden	# keine vorhanden	# keine vorhanden

Anlagen, Teil 2

Die nachfolgenden Abbildungen zeigen weiter Use Case Fälle der autosomalen Datenbank.



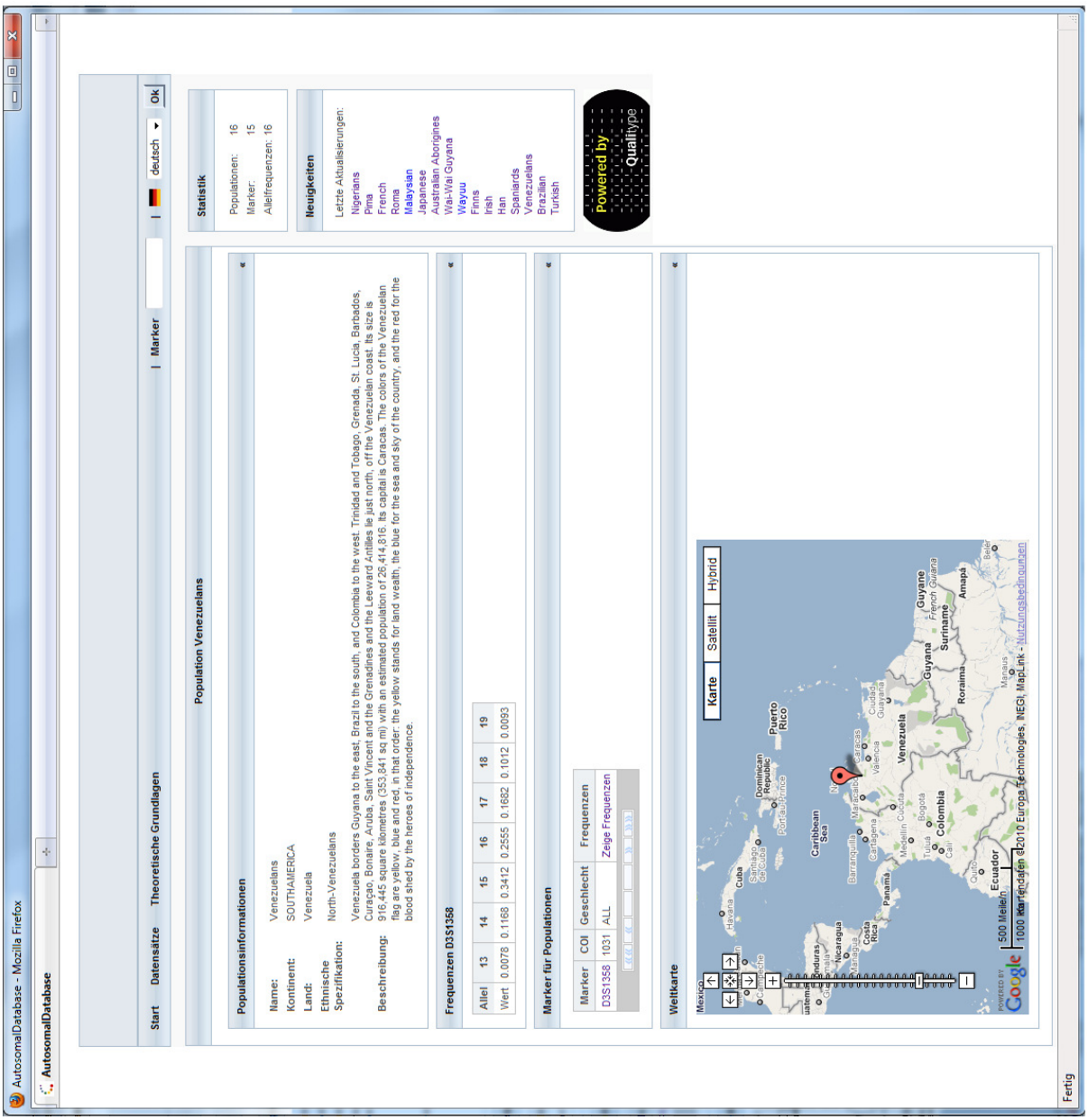


Anlagen, Teil 3

Das nachfolgende Diagramm zeigt das Klassenmodell der autosomalen Datenbank.

Anlagen, Teil 4

Die folgenden Darstellungen zeigen Web-Browser-Ausschnitte von verschiedenen Webseiten der autosomalen Datenbank.


AutosomalDatabase - Mozilla Firefox
AutosomalDatabase
Start
Datensätze
Theoretische Grundlagen
Marker
deutsch
Ok
Population Venezuelans
Populationsinformationen
Name: Venezuelans
Kontinent: SOUTHAMERICA
Land: Venezuela
Ethnische Spezifikation: North-Venezuelans
Beschreibung: Venezuela borders Guyana to the east, Brazil to the south, and Colombia to the west. Trinidad and Tobago, Grenada, St. Lucia, Barbados, Curacao, Bonaire, Aruba, Saint Vincent and the Grenadines and the Leeward Antilles lie just north, off the Venezuelan coast. Its size is 916,445 square kilometres (353,841 sq mi) with an estimated population of 26,414,816. Its capital is Caracas. The colors of the Venezuelan flag are yellow, blue and red, in that order: the yellow stands for land wealth, the blue for the sea and sky of the country, and the red for the blood shed by the heroes of independence.
Frequenzen D3S1358
Allel 13 14 15 16 17 18 19
Wert 0.0078 0.1168 0.3412 0.2555 0.1682 0.1012 0.0093
Marker für Populationen
Marker COI Geschlecht Frequenzen
D3S1358 1031 ALL Zeige Frequenzen
Weltkarte
Statistik
Populationen: 16
Marker: 15
Allelfrequenzen: 16
Neuigkeiten
Letzte Aktualisierungen:
Nigerians
Pima
French
Roma
Malaysian
Japanese
Australian Aborigines
Wai-Wai Guyana
Wayuu
Finns
Irish
Han
Spaniards
Venezuelans
Brazilian
Turkish
Powered by Qualitype
Karte
Satellit
Hybrid
Mexico
Cuba
Havana
Dominican Republic
Puerto Rico
Caribbean Sea
Venezuela
Colombia
Ecuador
Panama
Costa Rica
Nicaragua
Guyana
Suriname
Roraima
Amapá
Manaus
500 Meilen
1000 Kilometer
©2010 Europa Technologies, NEGI, MapLink - Nutzungsbedingungen
Fertig

Anlagen, Teil 5

Die nachfolgenden Abbildungen stammen aus einer wissenschaftlichen Publikation. Es wurden neun Marker einer türkischstämmigen Population untersucht.

RESEARCH NOTE

Short tandem repeat (STR) polymorphisms in Turkish population

ÜNER ÜLKÜER[1]*, MELAHAT KURTULUŞ-ÜLKÜER[2], CÜNEYT ELMA[1], TAHSİN KESİCİ[3] and SEVDA MENEVŞE[4]

[1]*General Directory of Security, Criminal Police Laboratories, Biyoloji Bölümü, Aniteppe, Ankara, Turkey*
[2]*Gazi University, Kirşehir Education Faculty, Kirşehir, Turkey*
[3]*Ankara University, Faculty of Agriculture, Biometry-Genetics Unit, Ankara, Turkey*
[4]*Gazi University, Faculty of Medicine, Department of Medical Biology and Genetics, Ankara, Turkey*

Introduction

In recent years, short tandem repeat (STR) systems have gained importance in forensic analysis of biological specimens as well as in paternity testing, as an alternative to the use of restriction fragment length polymorphism (RFLP) analysis (Edwards *et al.* 1991; Hammond *et al.* 1994; Nakamura *et al.* 1987). The analysis of STR polymorphisms by PCR-based method offers certain advantages over RFLP typing: (1) STR loci can be typed with a high degree of specificity and sensitivity in a short time period, (2) these loci can be succesfully amplified from a limited amount of DNA even if it is degraded, and (3) typing of multiple loci can be accomplished in a single multiplex reaction (Hochmeister *et al.* 1991; Lins *et al.* 1996). In this study, allele frequencies for the nine STR loci—D3S1358, vWA, FGA, D8S1179, D21S11, D18S51, D5S818, D13S317 and D7S820—were analyzed in 200 unrelated Turkish individuals. All loci, except for the FGA locus, were in Hardy-Weinberg equilibrium. The observed heterozygosity (H_o), expected heterozygosity (H_e), power of discrimination (PD), probability of exclusion (PE) and polymorphism information content (PIC) were calculated for the nine loci. The combined power of discrimination and combined probability of exclusion were 0.999999 and 0.999909 over the nine STR loci, respectively. These results suggest that these nine STR loci are a useful marker for forensic identification and paternity analysis.

Materials and methods

Sample preparation

This study was carried out on Turkish individuals living in Turkey, and randomly chosen from criminal cases. The samples were collected from 200 unrelated Turkish individuals. Genomic DNA was extracted from blood by the Chelex method (Walsh *et al.* 1991). The results obtained have previously been used for forensic purposes.

STR amplification and typing

1-5 ng of template DNA was used in each PCR reaction. Amplification of the STR loci were done using the AmpF/STR™ Profiler Plus Kit (Applied Biosystems, Foster City, CA, USA). AmpF/STR Profiler Plus PCR products were analyzed on ABI Prism™ 377 DNA sequencer. The GeneScan-500 (ROX) internal lane size standard was used.

Statistical analysis

The exact test was used to verify whether the genotypic distribution at each locus was in conformity with Hardy-Weinberg equilibrium (Guo *et al.* 1992). The power of discrimination (PD), probability of paternity exclusion (PE), polymorphism information content (PIC), expected heterozygosity (H_e) and observed heterozygosity (H_o) were also calculated (Fisher *et al.* 1951; Nei 1978; Botstein *et al.* 1980; Chakravarti *et al.* 1983).

Results and discussion

The observed allele frequencies of the nine STR loci are shown in table 1. The exact test was applied to these frequencies to test whether these loci taken into consideration were in Hardy-Weinberg equilibrium. The results indicated that the loci were in Hardy-Weinberg equilibrium, except for the FGA locus ($p = 0.027$), a deviation that may perhaps be due to migration. The expected (H_e) and observed (H_o) heterozygosity, PD, PE, and PIC values are presented in table 2. The heterozygosity of the nine STR loci screened in this study ranged from 0.735 to 0.860 (table 2), indicating that these loci could be used in determination of identity because of the high heterozygosity. These loci can also

* For correspondence. E-mail: unerulkuer@hotmail.com

Keywords. forensic science; population genetics; short tandem repeat; polymorphism; human genetics; Turkey.

Table 1. Allele frequencies at nine STR loci in Turkish population.

Allele	D3S1358	vWA	FGA	D8S1179	21S11	D18S51	D5S818	D13S317	D7S820
7							0.002		0.005
8				0.015			0.007	0.130	0.162
9				0.015		0.002	0.052	0.070	0.090
10				0.062		0.007	0.107	0.082	0.250
11				0.050		0.020	0.340	0.305	0.257
12	0.005			0.110		0.097	0.320	0.275	0.162
13	0.002	0.002		0.280		0.147	0.162	0.100	0.050
13.2						0.020			
14	0.072	0.072		0.222		0.137	0.007	0.035	0.017
14.2						0.027			
15	0.250	0.122		0.147		0.125		0.002	0.005
16	0.337	0.195		0.072		0.157			
17	0.180	0.302		0.020		0.090			
18	0.137	0.202	0.005	0.005		0.065			
19	0.015	0.087	0.040			0.047			
20		0.015	0.090			0.030			
21			0.190			0.015			
22			0.140			0.010			
23			0.205						
24			0.170						
25			0.110						
26			0.040		0.002				
26.2									
27			0.007		0.015				
28			0.002		0.127				
29					0.222				
29.2					0.002				
30					0.235				
30.2					0.022				
31					0.055				
31.2					0.132				
32					0.012				
32.2					0.110				
33					0.002				
33.2					0.045				
34					0.002				
34.2					0.010				
35					0.002				

accurately distinguish between two unrelated people since the discrimination power of these loci was very high (combined PD = 0.999999; combined PE = 0.999909). As seen in table 2, the PE and PIC of the D18S51 locus (PE = 0.782; PIC = 0.880) were higher than the others, whereas the PE value of the D5S818 locus was the lowest (PE = 0.512; PIC = 0.699).

The combined PD, PE and PIC values of nine loci in Turkish population were also compared with those from Japanese, Spanish, French-Canadian Caucasian and Greek populations. The combined PE value (0.9999) of Turks was slightly higher than that of Japanese (0.9991), African-American 0.9996) and U.S. Caucasian (0.9994), Spanish population (0.9998) and French-Canadian Caucasian population (0.9874) (data for the non-Turkish populations are from Yamamoto *et al.* 1999; Arce *et al.* 2001; Busque *et al.* 1997). The combined PD value in the Turkish population (0.999999) also seems to be a little higher than that of French-Canadian Caucasian population (0.999998) (Busque *et al.* 1997). However, the combined PD value of the Turkish population was similar to Japanese (0.999999), African-American (0.999999) and U.S. Caucasian (0.999999), Spanish population (0.999999) (Yamamoto *et al.* 1999; Arce *et al.* 2001). The H_e, H_o, PE, PD and PIC values calculated in this study were compared with those reported for a Greek population (Skitsa *et al.* 2003) and were found to be generally similar, although slight differences were observed at some loci for H_o and PE values. It can be concluded that the nine STR loci studied here appear to be informative genetic markers for identity and paternity testing in the Turkish population.

Acknowledgements

The second author would like to thank Prof. Dr. Zahide Kocabaş for helpful comments on the paper.

Table 2. Result of exact test for Hardy-Weinberg equilibrium, and statistical properties of nine STR loci in Turkish population.

Locus	D3S1358	vWA	FGA	D8S1179	D21S11	D18S51	D5S818	D13S317	D7S820
p	0.200	0.374	0.027*	0.194	0.079	0.094	0.141	0.583	0.065
H_e	0.769	0.803	0.852	0.828	0.845	0.894	0.743	0.793	0.809
H_o	0.805	0.770	0.800	0.790	0.820	0.860	0.735	0.805	0.770
PD	0.909	0.932	0.959	0.948	0.957	0.979	0.891	0.928	0.935
PE	0.551	0.612	0.698	0.659	0.689	0.782	0.512	0.599	0.619
PIC	0.731	0.774	0.832	0.804	0.825	0.880	0.699	0.763	0.780

P, exact test P-values; * $P < 0.05$; H_e expected heterozygosity; H_o observed heterozygosity; PD power of discrimination; PE probability of exclusion; PIC polymorphism information content.

References

Arce B., Heinrichs B., Armenteros M. F., Carrasco F., Lorente J. A. and Budowle B. 2001 Spanish population data on nine STR loci. *J. Forensic Sci.* **46,** 1003–1004.

Botstein D., White R. L., Skolnick M. and Davis R. W. 1980 Construction of a genetic linkage map in man using restriction fragment length polymorphisms. *Am. J. Hum. Genet.* **32,** 314–331.

Busque L., Desmarais D., Provost S., Schumm J. W., Zhong Y. and Chakraborty R. 1997 Analysis of allele distribution for six short tandem repeat loci in the French Canadian population of Quebec. *J. Forensic Sci.* **42,** 1147–1153.

Chakravarti A. and Li C. C. 1983 The effect of linkage on paternity calculations. In *Inclusion probabilities in parentage testing* (ed. R. H. Walker), pp. 411–422. American Association of Blood Banks, Arlington, USA.

Edwards A., Civitello A., Hammond H. A. and Caskey C. T. 1991 DNA typing and genetic mapping with trimeric and tetrameric tandem repeats. *Am. J. Hum. Genet.* **49,** 746–56.

Fisher R. A. 1951 Standard calculations for evaluating a blood-group system. *Heredity* **5,** 95–102.

Guo S. W. and Thompson E. A. 1992 Performing the exact test of Hardy-Weinberg proportion for multiple alleles. *Biometrics* **48,** 361–372.

Hammond H. A., Jin L., Zhong Y., Caskey C. T. and Chakraborty R. 1994 Evaluation of 13 short tandem repeat loci for use in personal identification applications. *Am. J. Hum. Genet.* **55,** 175–89.

Hochmeister M. N., Budowle B., Jung J., Borer U. V., Comey C.T. and Dirnhofer R. 1991 PCR-based typing of DNA extracted from cigarette butts. *Int. J. Leg. Med.* **10,** 506–13.

Lins A. M., Sprecher C. J., Puers C. and Schumm J. W. 1996 Multiplex sets for amplification of polymorphic short tandem repeat loci- silver stain and fluorescent detection. *BioTechniques* **20,** 882–889.

Nakamura Y., Leppert M., O'Connell P., Wolff R. and Holm T. 1987 Variable number of repeat (VNTR) markers for human gene mapping. *Science* **235,** 1616–1622.

Nei M. 1978 Estimation of average heterozygosity and genetic distance from a small number of individuals. *Genetics* **89,** 583–590.

Perkin Elmer Applied Biosystems 1998 *AmpFl STR Profiler Plus ™ User's Manual*. Foster City, USA.

Skitsa I., Salas A., Lareu M. V. and Carracedo A. 2003 STR-CODIS typing in Greece. *Forensic Sci. Int.* **137,** 104–106.

Yamamoto T., Uchihi R., Nozawa H., Huang X. L., Leong Y. K., Tanaka M. *et al.* 1999 Allele distribution at nine STR loci-D3S1358, vWA, FGA, TH01, TPOX, CSF1PO, D5S818, D13S317 and D7S820-in the Japanese population by multiplex PCR and capillary electrophoresis. *J. Forensic Sci.* **44,** 167–170.

Walsh P. S., Metzger D. A. and Higuchi R. 1991 Chelex 100 as a medium for simple extraction of DNA for PCR-based typing from forensic material. *BioTechniques* **10,** 506–13.

Received 26 June, 2003; in revised form 30 April, 2004

Eidesstaatliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.
Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.
Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Mittweida, den 31. Oktober 2010

Alexander Teicher